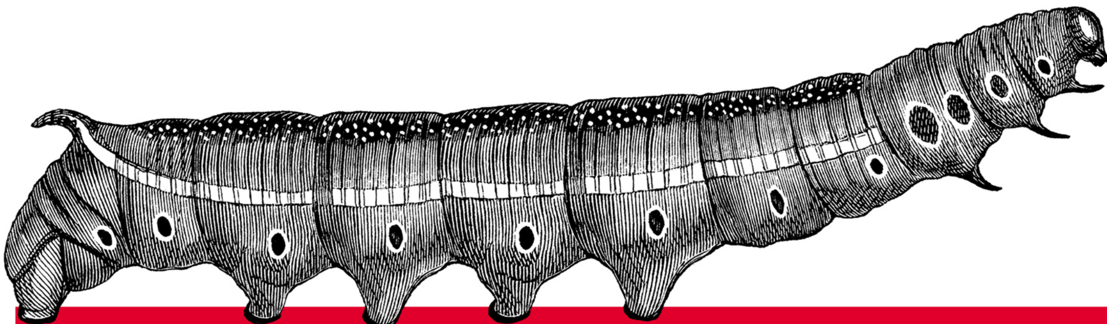
O'REILLY®

TURING 图灵程序设计丛书



Python 数据分析基础

Foundations for Analytics with Python

零编程经验也可学会用最火的Python语言进行数据分析

[美] Clinton W. Brownley 著

陈光欣 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

译者介绍

陈光欣

毕业于清华大学并留校工作，主要兴趣为数据分析与数据挖掘。

TURING 图灵程序设计丛书

Python数据分析基础

Foundations for Analytics with Python

[美] Clinton W. Brownley 著
陈光欣 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北　　京

图书在版编目（CIP）数据

Python数据分析基础 / (美) 克林顿·布朗利 (Clinton W. Brownley) 著 ; 陈光欣译. -- 北京 : 人民邮电出版社, 2017.8
（图灵程序设计丛书）
ISBN 978-7-115-46335-7

Ⅰ. ①P… Ⅱ. ①克… ②陈… Ⅲ. ①软件工具－程序设计 Ⅳ. ①TP311.561

中国版本图书馆CIP数据核字(2017)第165176号

内 容 提 要

本书展示如何用 Python 程序将不同格式的数据处理和分析任务规模化和自动化。主要内容包括：Python 基础知识介绍、CSV 文件和 Excel 文件读写、数据库的操作、示例程序演示、图表的创建，等等。

本书适合数据分析与处理工作相关人员。

◆ 著　[美] Clinton W. Brownley
译　陈光欣
责任编辑　朱　巍
执行编辑　张海艳
责任印制　彭志环

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：17
字数：402 千字　2017年 8 月第 1 版
印数：1－4 000册　2017年 8 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-4510号

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

献给 Aisha 和 Amaya，

“教育不是把桶灌满，而是将火点燃。”——苏格拉底

愿你们心中之火永远不熄。

目录

前言

本书面向的读者是那些经常使用电子表格软件进行数据处理，但从未写过一行代码的人。前几章会教你设置 Python 运行环境，告诉你计算机是如何看待数据并对其进行简单处理的。你很快就能掌握在电子表格（包括 CSV 文件）和数据库中处理数据的方法。

刚开始，你可能会觉得这样做是一种退步，如果你能熟练使用 Excel，这种感受会更加强烈。以前你只需复制粘贴就能完成的工作，现在却要煞费苦心地告诉 Python 如何在列的每个单元格之间循环，这效率太低了，想想就令人沮丧（特别是当你几次三番地回头去找某一处输入错误的时候）。但是当你逐渐掌握了 Python 之后，就会不断地发现它的真正价值所在，而其中一个极好的例子就是它可以自动完成你现在不断重复的工作。

本书的写作目的是让你全面地掌握 Python，然后充满信心地写出按照你的期望运行的有效代码。一开始输入一些代码或许是个好主意，这样你就会熟悉像制表符、闭括号和引用之类的技术细节。但是，本书中的所有代码在网上都能找到（https://github.com/cbrownley/foundations-for-analytics-with-python）。你在做自己的工作时，完全可以通过复制粘贴来重用这些代码。没关系！适时地进行复制和粘贴也是高效编程的一部分。在阅读本书的同时完成示例程序，会使你更好地理解示例代码的原理。

祝你在成为程序员的道路上好运连连！

为什么要读这本书，为什么要学习这些技能

如果你经常做数据处理工作，就一定会为学习编程而兴奋。学习编程的一个好处是，你可以完成那些靠手工难以完成或者根本不可能完成的数据处理与分析工作。可能你已经遇到了这样的问题：需要处理的文件包含太多数据，以至于打开文件都非常困难或者根本不可行。即使打开了这些文件，手动处理也会花费大量时间，并且极易出错，因为你对数据进行的任何修改都需要很长时间才能更新，而且面对如此多的数据，进行修改时很容易漏掉某一行或某一列。你可能还遇到了其他情况，如需要处理大量的文件，以至于手动处理根本不可能完成。有些时候，你需要的数据来自于几十、几百甚至上千个文件。当所需的文件数量不断增加时，手动处理会变得越来越困难。在以上所有这些情况之下，写一个

Python 脚本来处理文件就可以解决你的问题，因为 Python 脚本可以快速有效地处理大型文件和大批量的文件。

学习编程的另一个好处是，你可以自动地重复数据处理和数据分析过程。在很多情况下，我们针对数据做的都是耗时的重复性工作。例如，一般的数据管理过程是，先从客户或供应商处获取数据，然后提取并保留所需的数据，之后还可能会进行一些数据转换或重新格式化，最后将数据保存到数据库或数据仓库中［这就是数据科学家熟知的数据 ETL（extract、transform、load，即抽取、转换和加载）过程］。类似地，典型的数据分析过程包括数据获取、数据准备、数据分析和结果展示。在数据管理和数据分析过程中，一旦建立了流程，就可以编写 Python 代码来进行各种操作。通过创建 Python 脚本来执行操作，你可以将耗时的重复性工作简化为执行一个脚本，并用节省下来的时间去做其他更有意义的工作。

最重要的是，在进行数据处理和数据分析时，使用 Python 脚本代替手动操作可以减小出错的可能性。手动进行数据处理时，非常可能出现复制粘贴错误或输入错误。导致出错的原因有很多：你可能因过于匆忙而忽略了错误，或者有些事导致你分心了，或者仅是因为你太累了。而且，当你处理大型文件或大批量的文件，或者进行重复性操作时，出错的可能性会更大。相反，Python 脚本从来不会分心或疲劳。一旦你调试好脚本，确认它可以按照你的期望处理数据，它就会一如既往、不知疲倦地工作下去。

最后，学习编程非常有趣，而且能提高自身能力。只要熟悉了基本的语法，你就会非常乐于找到所需的语言功能，然后将它们组合在一起，以完成整体的数据分析目标。至于代码和语法，网上有许多示例可以教会你如何使用专门的功能来完成特定的任务。不过，这些示例虽能提供帮助，但是你需要通过自己的创造力和解决问题的能力来弄清楚如何修改这些代码，以使它们满足你的实际需要。找到合适的代码，并想办法让它们为你工作，这是个非常有意思的过程。此外，学习编程能极大地提高自身的能力。举个例子，考虑一下我前面提到过的情况，即要处理大型文件和大批量文件。如果不会编程，那么你要么需要花费大量时间，要么束手无策。一旦学会了编程，你就可以通过 Python 脚本轻松愉快地解决所有问题。有些数据处理和数据分析任务以前是非常困难或根本不可能完成的，但是现在你都可以轻松搞定，这会使你充满信心，能量爆棚，从而积极主动地寻找更多的机会，使用 Python 来迎接数据处理方面的挑战。

目标读者

本书的目标读者是那些经常从事数据处理工作，又具有极少或根本没有编程经验的人。书中的示例覆盖了常用的数据源和数据格式，包括文本文件、逗号分隔值（CSV）文件、Excel 文件和数据库。在某些情况下，由于文件中数据过多，或由于文件数量太多，造成文件难以打开或不能通过手动处理。在其他一些情况下，从文件中抽取和使用数据的过程非常耗时并且容易出错。在这些情况下，如果你不会编程，就会将大量时间浪费在数据搜索、打开与关闭文件，以及复制和粘贴数据上面。

鉴于你可能从未运行过脚本，本书从最基本的操作开始，介绍如何在文本文件中编写代码以创建 Python 脚本。然后，我们会学习如何通过命令行窗口（Windows 用户）或终端窗口（macOS 用户）来运行 Python 脚本。（如果你做过一点编程，可以跳过第 1 章，直接学习第 2 章中的数据分析内容。）

本书的编写方式特别适合编程新手。书中提供的示例包含了完成某项任务所需的全部 Python 代码，而不是仅提供一些代码片段，让你自己将它们组合起来以完成任务。你以后可能会经常使用本书作为参考，而且会发现书中的代码确实有帮助。最后，正所谓“一图胜千言”，书中使用了大量屏幕截图来展示输入文件、Python 脚本、命令行窗口、终端窗口和输出文件，这样你就可以真实地看到如何创建输入、代码、命令和输出了。

我会详细讲解代码的原理，也会推荐一些工具供你使用。这种方法可以帮助你打下坚实的基础，以理解在程序背后到底发生了什么。有时候，你需要在 Google 上搜索问题的解决方案并找到一些有用的代码。做完了书中的练习之后，你可以更好地理解这些代码的工作原理，也就是说，你不但知道如何根据具体情况使用它们，而且知道在出现问题时如何进行修复。因为你在每一章中都会编写一些代码，所以你会发现可以将本书作为参考书，或指导手册，然后在里面找到完成具体任务的方法。但是请记住，这仅是一本“学习如何编程”的书，你还需要不断提高和扩展编程技能，以便综合运用它们来完成各种任务。

为什么使用Windows

本书中的大部分示例都是演示在 Microsoft Windows 系统下如何创建和运行 Python 脚本。将重点放在 Windows 系统上的原因很简单：我想让本书帮助尽可能多的人。根据估计，大多数台式机和笔记本电脑（特别是用于商业分析的计算机）运行的是 Windows 操作系统。例如，根据 Net Applications 的调查，截至 2014 年 12 月，Microsoft Windows 占领了大约 90% 的台式机和笔记本电脑操作系统市场。因为我想让本书满足台式机用户和笔记本电脑用户的需求，而且这些电脑中多数都安装了 Windows 操作系统，所以本书将集中讲述如何在 Windows 系统下创建和运行 Python 脚本。

尽管本书将重点放在了 Windows 上，但在适当情况下，我也提供如何在 macOS 系统上创建和运行 Python 脚本的示例。不论在哪种机器上运行，Python 中几乎所有功能的表现都是一样的。当因为操作系统不同而出现差别时，我会分别给出具体的说明。例如，第 1 章的第一个例子演示了如何在 Microsoft Windows 和 macOS 系统下创建和运行 Python 脚本。类似地，第 2 章和第 3 章的第一个例子也演示了如何在 Windows 和 macOS 系统下创建和运行 Python 脚本。此外，第 8 章涵盖了两种操作系统，介绍了如何在 Windows 中建立计划任务以及如何在 macOS 中建立定时作业。如果你是 Mac 用户，可以使用每章的第一个例子作为模板，来学习如何创建 Python 脚本，如何使其可以执行，以及如何运行脚本，然后重复这些步骤来创建和运行每章中其余的示例程序。

为什么使用Python

如果你的目的是学习一门编程语言来使数据处理和数据分析任务规模化和自动化，那么 Python 绝对是一个好的选择。Python 的一个显著特点就是使用空白字符和缩进来表示行的结尾和代码分块，这与很多其他语言不同，其他语言使用特殊字符（比如分号和花括号）来达到同样的目的。Python 的这个特点使你一眼就能看出程序的组织方式。

在其他语言中，特殊字符的使用对于编程新手来说是个困扰，原因至少有两个。第一，这使得学习曲线更长并且更加陡峭。当你学习编程时，实质上是在学习一门新的语言，你必

须花时间学习这些特殊字符的用法，然后才能有效地使用这门语言。第二，特殊字符使代码难以阅读。这是因为在使用分号和花括号表示代码块的语言中，并不总是使用缩进来标明代码块。如果没有缩进，多个代码块看上去就是乱七八糟的。

Python 使用空白字符和缩进来表示代码分块，而不使用分号和花括号，这样就避免了上述问题。当你阅读 Python 代码时，你的视线会集中在实际的代码行上，而不是代码块的分隔符上，因为代码周围只有空白字符。Python 要求代码块必须缩进，这样你会很容易看出代码块在哪里结束，新的代码块又在哪里开始。而且，Python 社区特别强调代码的可读性，因此已经形成了一种文化，就是一定要书写易于阅读和理解的代码。Python 的这些特点使学习曲线更短并且更加平坦，与其他语言相比，使用 Python 进行数据处理可以更快也更容易上手。

Python 适用于数据处理与分析的另一个显著特点，是其具有大量的标准模块、附加模块以及函数，可以非常方便地完成一般的数据处理与分析操作。内建库和标准库中的模块和函数是 Python 的标准配置，所以只要你下载并安装了 Python，就可以立即使用这些内建的模块和函数。在 Python 标准库页面（https://docs.python.org/3/library）中，你可以找到所有内建模块和标准模块的介绍。Python 附加模块需要单独下载并安装，然后才能使用它们提供的附加功能。你可以在 Python 程序包索引页面（https://pypi.python.org/pypi）详细查看很多附加模块的介绍。

标准库中的模块提供的功能包括读取各种类型的文件（如文本文件、CSV、JSON、HTML、XML 等），处理数值、字符串和日期型数据，使用正则表达式进行模式匹配，解析 CSV 文件，计算基本的统计量，以及向各种类型的输出文件和磁盘写入数据。有用的附加模块太多，无法一一介绍。本书要讨论和使用的附加模块如下所示。

- `xlrd` 和 `xlwt`

  功能：解析与读写 Microsoft Excel 工作簿。

- `mysqlclient/MySQL-python/MySQLdb`

  功能：连接 MySQL 数据库，在数据库表上运行查询。

- `pandas`

  功能：读取各种类型的文件；管理、筛选和转换数据；聚合数据并计算基本统计量；创建各种类型的统计图表。

- `statsmodels`

  功能：估计各种统计模型，包括线性回归模型、广义线性模型和分类模型。

- `scikit-learn`

  功能：估计机器学习统计模型，包括回归、分类和聚类，以及执行数据处理、维度归约和交叉验证。

如果你是编程新手，并且正在寻找一门可以使数据处理与分析任务自动化和规模化的编程语言，那么 Python 就是理想的选择。Python 对于空白字符和缩进的强调使代码更易于阅读和理解，因而和其他语言相比，它的学习曲线没有那么陡峭。Python 的内建库和附加库可以方便地完成许多一般的数据处理和分析操作，让你可以轻松地一站式完成数据处理与分析任务。

基础Python和pandas

pandas 是 Python 的一个功能强大的附加模块，提供对数据进行读 / 写、组合、转换和管理的功能，同时还可以计算统计量并创建统计图表。在完成数据处理任务时，使用 pandas 提供的功能可以大大减轻你的编码工作量。这个模块深受数据分析师和其他 Python 使用者青睐，因为它的功能实用且强大，运行速度快，使用简单，能够减少完成任务所需的代码量。正因为 pandas 功能强大且深受欢迎，所以本书将向你介绍它。本书在第 2 章和第 3 章中提供 pandas 版的 Python 脚本，在第 6 章中介绍如何使用 pandas 创建统计图表，在第 7 章中演示如何通过 pandas 计算各种统计量。我建议你学习一下 Wes McKinney 的著作《利用 Python 进行数据分析》[1]。

同时，如果你是编程新手，我还建议你学习一下基本的编程技能。一旦具有了编程技能，你就可以扩展自己解决问题的能力。你可以将复杂的问题分解成几个较小的问题，分别解决，然后将它们组合起来解决更大的问题。你还可以培养出一种直觉，去决定使用哪种数据结构和算法来有效且高效地解决各种问题。此外，你还会遇到像 pandas 这样的附加模块无法解决或者无法以你需要的方式解决的问题。在这种情况下，如果你没有基本的编程技能，就会束手无策。相反，如果你具有了编程技能，就可以创建所需功能，从而独立解决问题。能够独立解决编程问题是非常令人振奋的，并能极大提升个人能力。

因为本书是面向编程新手的，所以我会将重点放在基础的、通用的编程技能上。比如，第 1 章介绍数据类型、数据容器、控制流、函数、`if-else` 逻辑和文件读写等基本概念。此外，在第 2 章和第 3 章中，每种脚本都提供两个版本的实现方式：基础 Python 版和 pandas 版。在每个案例中，我会首先讨论基础 Python 版的实现，让你学会独立编码解决问题，然后再给出 pandas 版的实现。我希望你能够从基础 Python 版中学会基本的编程技能，这样在使用 pandas 版时，你就能够更加深刻地理解 pandas 简化了的概念和操作。

Anaconda Python

当开始使用 Python 时，有很多程序可以用来编写代码。例如，如果你从 Python.org 下载了 Python，在安装完成之后，就会得到一个具有图形用户界面（GUI）的文本编辑器，它叫作 Idle。另外，你可以下载 IPython Notebook，在一种基于 Web 的交互式环境下编写代码。如果你在 macOS 系统下工作，或者在 Windows 系统上安装了 Cygwin，那么就可以在终端窗口中使用 Nano、Vim 或 Emacs 等内置文本编辑器编写代码。如果你已经熟悉了上面的任意一种程序，那么就可以随意使用它来处理本书中的示例代码。

但是，这里我要介绍如何从 Continuum Analytics 下载并安装免费的 Anaconda Python 发行版，因为与其他版本相比，它对于编程新手来说具有很多优点，而且同样适合编程老手！Anaconda Python 最主要的优点是，它会预先安装几百个最流行的 Python 附加模块，所以你无需一个一个地安装这些模块以及它们的依赖模块。举例来说，本书要使用的所有附加模块在 Anaconda Python 中都预先安装好了。

注 1：Wes McKinney 是 pandas 模块最初的开发者，他的这本书是学习 pandas、NumPy 和 IPython 的绝好教材，如果你想扩展一下使用 Python 进行数据分析的知识，这些是你应该学习的附加模块。

另一个优点是，它同时提供了一个名叫 Spyder 的集成开发环境（IDE）。Spyder 具有非常方便实用的界面，供你编写、运行和调试代码，还可以安装程序包和启动 IPython Notebook。另外，它还具有很多美妙的功能，比如在线文档链接、语法着色、键盘快捷方式和错误提示。

Anaconda Python 还有一个优点是跨平台性——具有 Linux、Mac 和 Windows 3 个版本。所以，如果你在 Windows 系统下熟悉了它的用法，在转到 Mac 系统时，仍然可以使用同样熟悉的界面。

如果你已经熟悉了 Python 和所有可用的附加程序包，那么在使用 Anaconda Python 时需要注意一点，就是安装附加程序包时的语法有些差别。在 Anaconda Python 中，你需要使用 `conda install` 命令。举例来说，要安装附加程序包 `argparse`，你应该输入 `conda intall argparse`。这种语法与通常的 `pip install` 是不同的。（如果你从 Python.org 下载并安装了 Python，那么安装 `argparse` 包应该使用 `python -m pip install argparse`。）Anaconda Python 也允许你使用 `pip install` 语法，所以实际上可以使用任何一种方式，但是当你学习如何安装附加程序包时，应该知道这一点微小的区别。

安装Anaconda Python（Windows或Mac）

要安装 Anaconda Python，需遵循以下步骤。

(1) 访问 http://continuum.io/downloads（网站会自动检测出你的操作系统，即 Windows 或 Mac）。

(2) 选择 Windows 64-bit Python 3.5 Graphical Installer（如果你使用 Windows）或者 Mac OS X 64-bit Python 3.5 Graphical Installer（如果你使用 Mac）。

(3) 双击已下载的 .exe 文件（Windows 系统）或 .pkg 文件（Mac 系统）。

(4) 按照安装程序的指示操作。

文本编辑器

尽管本书中会使用 Anaconda Python 和 Spyder，但是熟悉一下其他可用于编写 Python 代码的文本编辑器还是有意义的。例如，如果你不想使用 Anaconda Python，可以简单地从 Python.org 下载安装 Python，然后使用像 Notepad（Windows 系统）和 TextEdit（macOS 系统）这样的文本编辑器。要使用 TextEdit 编写 Python 脚本，你需要打开 TextEdit，将 TextEdit → Preferences 下面的单选按钮从“Rich text”改为“Plain text”，这样新文件就会以普通文本方式打开。然后你就可以使用扩展名 .py 保存文件了。

使用文本编辑器编写代码的好处是，它已经安装在你的计算机上了，所以你不用担心如何下载和安装一个新的软件。大多数台式机和笔记本电脑在出厂时都带有文本编辑器，如果你不得不使用一台没有 Spyder 或终端窗口的计算机，那就使用其自带的任意文本编辑器快速开始工作吧。

尽管完全可以使用像 Notepad 和 TextEdit 这样的文本编辑器编写 Python 代码，效率也很高，但是你还可以下载其他免费的文本编辑器，因为它们提供了一些额外的功能，包括代

码高亮显示、制表符长度调整，以及多行缩进与减少缩进。这些功能（特别是代码高亮显示和多行缩进与减少缩进）非常有用，尤其是在你学习如何编写和调试代码时。

下面是一个提供这些功能的免费文本编辑器的不完全列表：

- Notepad++（http://notepad-plus-plus.org，Windows）
- Sublime Text（http://www.sublimetext.com，Windows 和 Mac）
- jEdit（http://www.jedit.org，Windows 和 Mac）
- TextWrangler（http://www.barebones.com/products/textwrangler，Mac）

再说一遍，本书使用 Anaconda Python 和 Spyder，但是你可以随意使用一种文本编辑器处理示例代码。如果你下载了某种文本编辑器，请在网上查一下可以用来进行多行缩进与减少缩进的按键组合。当开始试着调试代码块时，这会使你轻松许多。

下载本书资料

本书中的所有 Python 脚本、输入文件和输出文件都可以在这个网址找到：https://github.com/cbrownley/foundations-for-analytics-with-python。

可以将整个文件夹下载到你的计算机上，不过点击文件名然后将脚本复制粘贴到你的文本编辑器中会更简单。（GitHub 是进行代码分享与协作的一个网站，非常适合跟踪项目的不同版本并管理协作过程，但是它的学习曲线相当陡峭。当你准备开始分享自己的代码和提交对他人代码的改进时，可以参考一下 Chad Thompson 的教程 *Learning Git*（http://shop.oreilly.com/product/110000769.do，Infinite Skills）。

各章内容简介

- 第 1 章　Python 基础

  这一章介绍如何创建和运行 Python 脚本。该章的重点在于 Python 的基本语法和元素，你需要了解它们，才能学习后面的章节。例如，讨论像数值和字符串这样的基本数据类型，以及如何对它们进行操作；介绍主要的数据容器（列表、元组和字典），以及使用它们存储和操作数据的方法；介绍如何处理日期型数据，因为商业分析中经常出现日期。另外，还会讨论一些编程概念，比如控制流、函数和异常，它们是在编码中体现业务逻辑以及优雅地进行错误处理的重要元素。最后将介绍如何使计算机读入一个和多个文本文件，并且写回到 CSV 格式的输出文件中。这些技术对于访问输入数据和保存特定的输出数据都是非常重要的，后续章节会更深入地讨论这些问题。

- 第 2 章　CSV 文件

  这一章介绍如何读写 CSV 文件。首先介绍在不使用 Python 内置的 `csv` 模块情况下“手动”解析 CSV 格式的输入文件的一个例子。随后说明这种解析方法的潜在问题，并通过一个示例说明使用 Python 的 `csv` 模块解析 CSV 文件如何能避免这些问题。然后讨论如何使用 3 种不同类型的条件逻辑从输入文件中筛选出特定的行，将它们写入 CSV 格式的输出文件。接着给出两种不同的方法，以筛选出特定的列，并将它们写入输出文件。在介绍了如何读取和解析单个 CSV 格式的输入文件后，进一步讨论如何读取和处

理多个 CSV 文件。这一节中的示例包括为每个输入文件提供摘要信息，从多个输入文件中连接数据，以及为每个输入文件计算基本的统计量。这一章最后将介绍两个不太常用的过程示例，包括选择一组连续的行和为数据集添加标题行。

- 第 3 章　Excel 文件

  这一章讨论如何使用可下载的扩展模块 `xlrd` 读取 Excel 工作簿。首先介绍一个 Excel 工作簿示例（也就是说明工作簿中包含多少个工作表，每个工作表的名称，每个工作表中行与列的数量）。因为 Excel 将日期保存为数值型数据，所以下一节介绍如何使用一系列函数将日期格式化，以使它们显示为日期形式而不是数值形式。然后，讨论如何使用 3 种不同类型的条件逻辑从单个工作表中筛选出特定的行，再将它们写入 CSV 格式的输出文件。在此之后，介绍两种不同的方式来筛选特定的列并写入输出文件。在介绍了如何读取和解析单个工作表之后，进一步讨论如何读取和处理工作簿中所有的或者一部分工作表。这几节中的示例程序展示了如何在工作表中筛选特定的行与列。在讨论了如何读取和分析单个工作簿中的任意数目的工作表之后，进一步讨论如何读取和处理多个工作簿。这一节中的示例程序包括为每个工作簿提供摘要信息，从多个工作簿中连接数据，以及为每个工作簿计算基本的统计量。这一章最后将介绍两个不太常用的过程示例，包括选择一组连续的行和为数据集添加标题行。

- 第 4 章　数据库

  这一章讨论如何在 Python 中执行基本的数据库操作。首先介绍如何使用 Python 内建的 `sqlite3` 模块，这样你就不需要安装任何额外的软件了。示例程序说明了如何执行最常用的数据库操作，包括创建数据库和数据表，从 CSV 格式的输入文件加载数据到数据库中的表，使用 CSV 格式的输入文件更新数据表中的记录，以及查询数据表。使用 `sqlite3` 模块时，数据库连接的细节和与 MySQL、PostgreSQL 和 Oracle 等其他数据库系统连接有轻微的差别。为了说明这种差别，这一章的第二部分演示了如何同 MySQL 数据库系统进行交互。如果你的计算机上没有 MySQL，那么需要先下载并安装。然后，具体操作的示例程序与 `sqlite3` 示例相对照，也包括创建数据库和数据表，从 CSV 格式的输入文件加载数据到数据库中的表，使用 CSV 格式的输入文件更新数据表中的记录，查询数据表，以及将查询结果写入 CSV 格式的输出文件。这一章两部分的示例合在一起，可以详细又完整地说明如何使用 Python 执行常用的数据库操作。

- 第 5 章　应用程序

  这一章包含 3 个示例程序，演示了如何综合使用前面几章介绍的技术解决 3 个不同的问题，它们代表了一些常见的数据处理与分析任务。第一个应用程序介绍了如何在大量的 Excel 与 CSV 文件中找到特定的记录。可以想象，用计算机查询记录比手动查询要高效得多，也有趣得多。打开、搜索和关闭大量文件绝对不是一件有趣的事情，文件的数量越多，完成任务的难度就越大。因为这个问题涉及搜索 CSV 和 Excel 文件，所以示例程序会使用第 2 章和第 3 章中介绍的很多内容。

  第二个应用程序介绍如何将数据通过分组或“装箱”划分到一个唯一的类别，并且为每个类别计算统计量。具体的例子就是对一个记录客户服务包购买的 CSV 文件［记录了客户在什么时间购买了特定的服务包（也就是铜牌服务包、银牌服务包和金牌服务包）］进行解析，然后将数据按客户姓名和服务包进行组织，通过相加计算出每个客户在每种

服务包上花费的时间。这个示例使用了两个内建模块，创建了一个函数并将数据存储在字典中。字典在第 1 章中进行了介绍，但在第 2 章、第 3 章和第 4 章中都没有使用过。这个程序还引入了一种新的技术：记录下你刚处理过的行和正在处理的行，然后根据这两行的值计算出统计量。这两种技术（通过字典来分组或装箱数据，以及记录当前行和前一行）都非常强大，让你能够处理很多和时间相关的数据分析任务。

第三个应用程序介绍如何解析文本文件，将数据分组或装箱划分类别，然后按类别计算统计量。具体的例子是解析 MySQL 错误日志文件，按照日期和错误信息组织数据，然后计算出每种错误信息在每一天出现的次数。这个示例回顾了如何解析文本文件，这种技术在第 1 章中简要介绍过。这个示例也展示了如何将信息分别存储在列表和字典中，以用来创建输出文件的标题行和数据行；它还可以帮你回忆一下通过基本字符串操作来解析文本文件的方法。同时，这也是使用嵌套字典来分组或装箱数据以划分类别的一个绝好示例。

- 第 6 章　图与图表

  在这一章中，你要学习如何使用 Python 创建常用的统计图和图表。你将使用 4 个制图库：`matplotlib`、`pandas`、`ggplot` 和 `seaborn`。首先使用 `matplotlib`，因为它历史悠久、资料丰富（实际上，`pandas` 和 `seaborn` 都是在 `matplotlib` 的基础上开发出来的）。`matplotlib` 一节介绍如何创建直方图、条形图、折线图、散点图和箱线图。`pandas` 一节讨论使用 `pandas` 简化语法来创建这些统计图的几种方式，并演示如何使用 `pandas` 创建统计图。`ggplot` 一节指出了这个库与 R 和图形语法在历史上的联系，并演示如何使用 `ggplot` 创建常用的统计图。最后，`seaborn` 一节讨论如何创建标准统计图，以及如何创建使用 `matplotlib` 难以创建的图表。

- 第 7 章　描述性统计与建模

  这一章讨论如何生成标准摘要统计量，以及如何使用 `pandas` 和 `statsmodels` 包估计回归模型与分类模型。`pandas` 中有计算集中趋势测度（例如：均值、中位数和众数）的函数，也有计算分散程度（例如：方差和标准差）的函数，还有进行数据分组的函数用于轻松计算这些统计量。`statsmodels` 包中的函数可以估计多种类型的回归和分类模型。这一章介绍了如何基于 `pandas` 数据框中的数据建立多元线性回归和逻辑斯蒂分类模型，以及如何使用模型为新的输入数据预测输出值。

- 第 8 章　按计划自动运行脚本

  这一章介绍如何在 Windows 和 macOS 系统上安排脚本定期自动运行。在这一章之前，脚本都是通过命令行方式手动运行的。在调试脚本和临时运行时，通过命令行手动运行脚本是非常方便的。但是，如果脚本需要定期运行（例如：每天、每周、每月或每个季度），或者需要定期运行很多脚本的话，手动运行就会非常麻烦。在 Windows 系统中，你可以创建任务计划来定期自动运行脚本。在 macOS 系统中，你需要创建定时任务，它可以实现同样的功能。这一章用若干屏幕截图展示了如何创建和运行任务计划和定时任务。通过安排脚本定期运行，你就不会忘记运行脚本，而且能够实现比通过命令行手动运行脚本更强大的功能。

- 第 9 章　从这里启航

  最后一章介绍 Python 中其他的内置和扩展模块以及函数，它们对于数据处理与数据分

析任务也是非常重要的。这一章还介绍了其他的数据结构。当你涉及本书之外的主题时，可能会遇到一些非常复杂的编程问题，而使用这些数据结构可以帮助你高效地解决问题。内置模块与函数是与 Python 安装程序捆绑在一起的，所以当你安装了 Python 之后，立刻就可以使用它们了。这一章讨论的内置模块包括 `collections`、`random`、`statistics`、`itertools` 和 `operator`，内置函数包括 `enumerate`、`filter`、`reduce` 和 `zip`。扩展模块没有包括在 Python 安装程序中，所以需要你单独下载并安装。这一章讨论的扩展模块包括 NumPy、SciPy 和 Scikit-Learn；另外还简单介绍了栈、队列、树和图等其他数据结构，来帮助你更加快速和高效地存储、处理和分析数据。

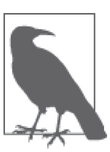
排版约定

本书使用了下列排版约定。

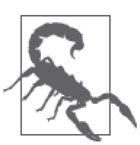
- **黑体**
  表示新术语或重点强调的内容。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 加粗等宽字体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。
- 等宽斜体（*`constant width italic`*）
  表示应该由用户输入的值或根据上下文确定的值替换的文本。



该图标表示提示或建议。



该图标表示一般注记。



该图标表示警告或警示。

使用示例代码

本书的所有补充资料（虚拟机、数据、脚本、定制的命令行工具等）都可以在这个地址下载：https://github.com/cbrownley/foundations-for-analytics-with-python。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*Foundations for Analytics with Python* by Clinton Brownley (O'Reilly). Copyright 2016 Clinton Brownley, 978-1-491-92253-8."

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online

Safari Books Online 是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商业作家的专业作品。

技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

对于本书的评论和技术性问题，请发送电子邮件到：bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：
http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia

本书作者的 Twitter 账号为：@ClintonBrownley

致谢

我写这本书的目的是帮助那些没有或只有一点编程经验的人，这些人和几年前的我很相似，我要帮助他们学习一些基本的编程技能，让他们因为能够完成数据处理和数据分析项目而欣喜若狂。在学习编程之前，这些项目可能相当耗时或者根本不可能完成。

如果不是很多人对我进行教导、指引和支持，我根本就不可能完成这本书。首先，我要感谢我的妻子 Anushka，她花了很多时间来教我基本的编程概念。她教我如何将复杂的编程任务分解为小任务，然后通过伪代码组织起来；教我如何有效地使用列表、字典和条件逻辑；还教我如何写出可扩展的、可伸缩的代码。开始时，她让我集中精力完成编程任务，不用太在乎代码是否优雅和高效。当我逐渐熟练之后，她总是会检查我的脚本并提出改进意见。在我写作本书期间，她一直这样支持着我。她检查了所有的脚本并提出了建议，使得脚本更加短小精悍、清晰易读，并且更加高效。她还做了大量的文字检查工作，并提出了增删和修改建议，以使得指令和解释更易于阅读和理解。在我写作的几个月中，除了这些教导和建议，她在其他方面也给了我极大的帮助。当我在夜里和周末写作时，她负责照顾我们的女儿；当我在写作中遇到困难时，她积极地鼓励我。没有她的指示、引导、批评、支持和始终不渝的爱，这本书根本不可能完成。

其次，我要感谢我的朋友和同事，他们鼓励、支持并帮助我进行编程训练。Heather Marquez 和 Ashish Kelkar 给予了我极大的支持，他们帮助我参加培训课程以及那些能增强和扩展编程技能的项目。当我告诉他们我编写了一套培训材料，想开设一门 10 天的培训课程之后，他们帮助我成功地实现了计划。Rajiv Krishnamurthy 也对我的学习帮助有加，一连几个星期，他为我提供各种各样的编程练习，并且每周和我碰头，讨论、评判和改进我的解决方案。Vikram Rao 审校了本书的线性回归和逻辑斯蒂回归部分，并针对如何说明回归模型的关键点提出了非常好的建议。我还要感谢其他很多同事，他们与我一起做项目，帮助我理解和掌握技术，与我共享代码，检查我的代码并提出改进意见，还为我提供有用的信息资源。

然后，我要感谢我的 3 位 Python 培训导师：Marilyn Davis、Jeremy Osborne 和 Jonathan Rocher。Marilyn 和 Jeremy 的课程讲授基本编程概念和 Python 实现。Jonathan 的课程讲授 Python 科学栈，包括 `numpy`、`scipy`、`matplotlib`、`seaborn`、`pandas` 和 `scikit-learn`。我真的非常喜欢他们的课程，他们每个人都扩展和丰富了我对基本编程概念和其 Python 实现的理解。

我还要感谢为这本书提供支持的 O’Reilly Media 的同仁。在本书的写作和编辑过程中，Timothy McGovern 一直是一位热情的伙伴。他审校了全部书稿，对本书的主题和各章内容都提出了极具价值的建议。他还对具体章节的文字、布局和格式提出了修改意见，使得它

们更加易于阅读和理解。感谢他的同事 Marie Beaugureau 和 Rita Scordamalgia，是她们与我一起完成了本书的出版过程并提供了市场资源。感谢 Colleen Cole 和 Jasmine Kwityn 编辑了所有章节并将整本书制作得如此精美。最后，感谢 Ted Kwartler 审校了本书的第一稿，并提出了有用的改善建议。他的意见促使我添加了可视化和统计分析章节，为每个基础 Python 脚本配备 pandas 版，并删除了一些内容和示例来减少重复以增强可读性。得益于他的深思熟虑，本书的内容才能更加丰富而全面。

电子书

扫描如下二维码，即可购买本书电子版。

第 1 章

Python基础

很多关于 Python 的图书和在线教程都展示了如何在 Python shell 中运行代码。要以这种形式运行 Python 代码，需要先打开一个命令行窗口（Windows 系统）或终端窗口（macOS 系统），输入“python”，按回车键之后会看见 Python 提示符（就是 >>>）。然后，只需一个一个地输入命令，Python 就会依次执行。

下面是两个典型的示例：

```
>>> 4 + 5
9

>>> print("I'm excited to learn Python.")
I'm excited to learn Python.
```

这种运行代码的方法简捷有趣，但是当代码的行数不断增加时，就不太合适了。当你的任务需要多行代码才能完成时，一种更简便的方式是将所有的代码写在一个称为 Python **脚本**的文本文件中，然后运行这个脚本。下面就说明创建 Python 脚本的方法。

1.1　创建Python脚本

要创建一个 Python 脚本，需执行下列步骤。

(1) 打开 Spyder IDE 或一个文本编辑器（例如：Windows 系统可以使用 Notepad、Notepad++ 或 Sublime Text；macOS 系统可以使用 TextMate、TextWrangler 或 Sublime Text）。

(2) 将下面两行代码写在文本文件中：

```
#!/usr/bin/env python3
print("Output #1: I'm excited to learn Python.")
```

第一行比较特殊，称为 shebang 行，在 Python 脚本中，你应该一直将它作为第一行。请注意行中的第一个字符是井号（#）。以 # 开头的行为单行注释，所以安装了 Windows 系统的计算机不读取也不执行这行代码。但是，安装了 Unix 系统的计算机使用这一行来找到执行文件中代码的 Python 版本。因为 Windows 系统忽略这一行，像 macOS 这样的基于 Unix 的系统使用这一行，所以加入这一行可以使脚本在不同操作系统的计算机之间具有可移植性。

第二行是一个简单的打印语句。这一行会将双引号之间的文本打印在命令行窗口（Windows）或终端窗口（macOS）上。

(3) 打开 Save As 对话框。
(4) 在 location 栏中切换到桌面，使文件可以保存到桌面上。
(5) 在 format 栏中，选择 All Files，使对话框不自动选择文件类型。
(6) 在 Save As 或 File Name 栏中，输入“first_script.py”。以前，你可能会将这个文本文件保存为 .txt 文件，但是在这个示例中，你应该把它保存为 .py 文件，来创建一个 Python 脚本。
(7) 点击 Save。

这样，你就创建了一个 Python 脚本。图 1-1、图 1-2 和图 1-3 分别展示了使用 Anaconda Spyder、Notepad++（Windows）和 TextWrangler（macOS）创建脚本的界面。

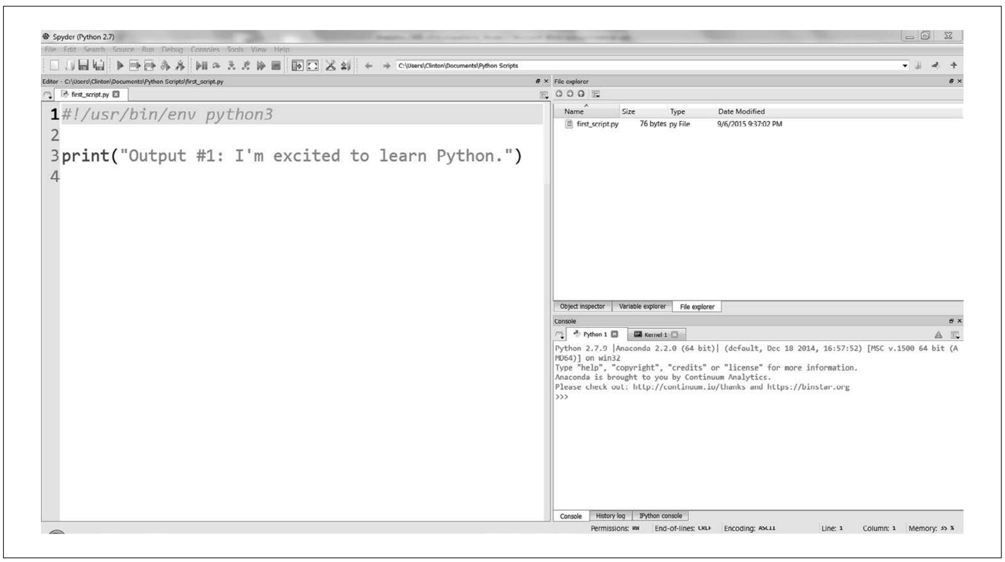


图 1-1：Anaconda Spyder 中的 Python 脚本 first_script.py

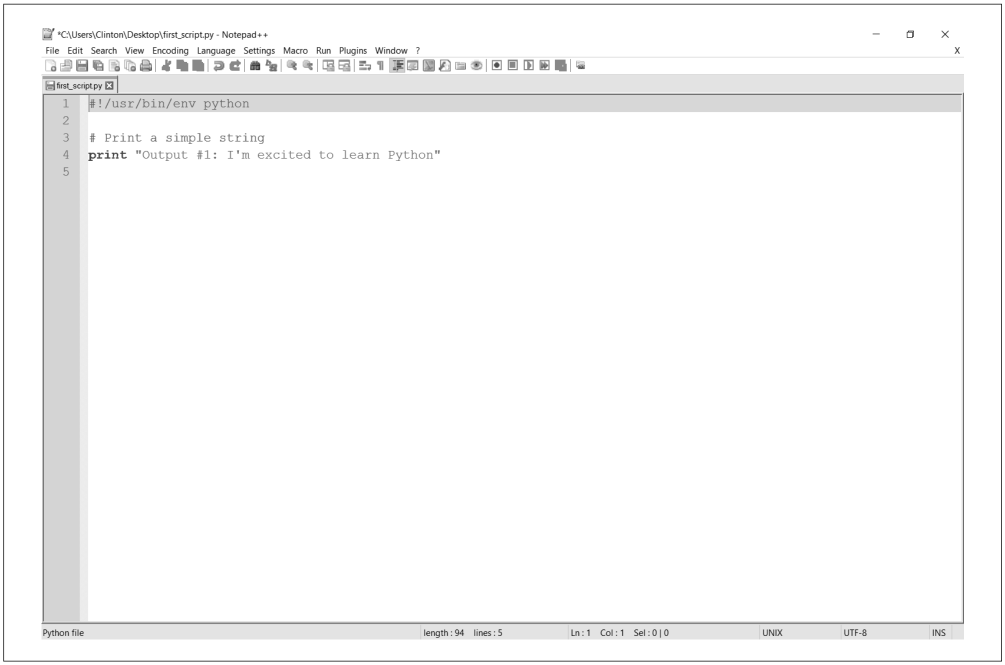


图 1-2：Notepad++（Windows）中的 Python 脚本



图 1-3：TextWrangler（macOS）中的 Python 脚本

下一节将介绍如何在命令行窗口或终端窗口中运行 Python 脚本。你会看到运行脚本和创建脚本一样简单。

1.2 运行Python脚本

如果你使用 Anaconda Spyder IDE 来创建脚本，那么点击 IDE 左上角的绿色三角（运行按钮）就可以运行脚本。

点击运行按钮之后，你就会看到输出显示在 IDE 右下窗格里面的 Python 控制台中。屏幕截图显示了绿色运行按钮和红框中的输出（参见图 1-4）。在这个示例中，输出为“Output #1: I'm excited to learn Python.”。

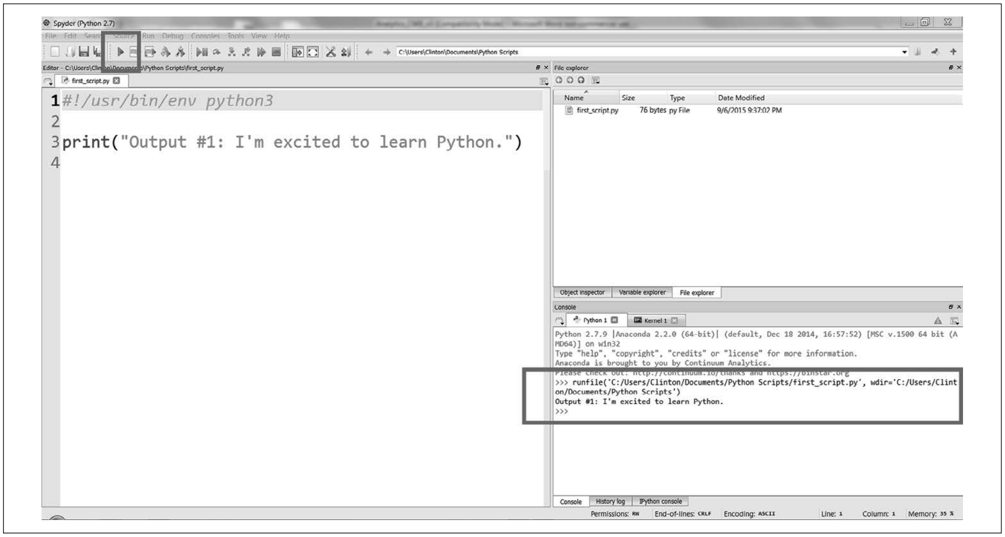


图 1-4：在 Anaconda Spyder 中运行 Python 脚本 first_script.py

或者，你可以在命令行窗口（Windows）或终端窗口（macOS）中运行脚本，如下所示。

- Windows 命令行窗口

  (1) 打开一个命令行窗口。

  当窗口打开后，提示符会是一个具体的文件夹，也称为目录（例如：C:\Users\Clinton 或 C:\Users\Clinton\Documents）。

  (2) 切换到桌面（将 Python 脚本保存在这里）。

  要完成这个操作，输入以下命令，然后按回车键：

```
cd "C:\Users\[Your Name]\Desktop"
```

  使用你的电脑账户名称，通常是你的名字，来替换 `[Your Name]`。例如，在我的电脑上，应该输入：

```
cd "C:\Users\Clinton\Desktop"
```

  这时候，提示符应该是这样的：`C:\Users\[Your Name]\Desktop`。这就对了，因为这就是保存 Python 脚本的地方。最后一步就是运行脚本了。

  (3) 运行 Python 脚本。

  要完成这个操作，输入以下命令，然后按回车键：

```
python first_script.py
```

你可以看到以下输出在命令行窗口中被打印出来，如图 1-5 所示。

```
Output #1: I'm excited to learn Python.
```

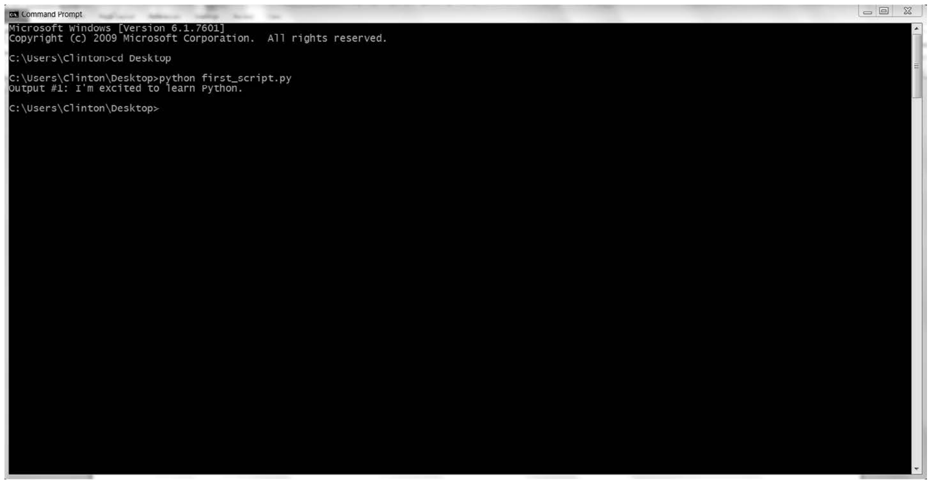


图 1-5：在命令行窗口（Windows）中运行 Python 脚本

- 终端窗口（Mac）

  (1) 打开一个终端窗口。

  当窗口打开后，提示符会是一个具体的文件夹，也称为目录（例如：/Users/clinton 或 /Users/clinton/Documents）。

  (2) 切换到桌面，将 Python 脚本保存在这里。

  要完成这个操作，输入以下命令，然后按回车键：

  ```
  cd /Users/[Your Name]/Desktop
  ```

  使用你的电脑账户名称，通常是你的名字，来替换 [*Your Name*]。例如，在我的电脑上，应该输入：

  ```
  cd /Users/clinton/Desktop
  ```

  这时候，提示符应该是这样的：/Users/[*Your Name*]/Desktop。这就对了，因为这就是保存 Python 脚本的地方。下一步是为脚本添加执行权限，然后运行脚本。

  (3) 为 Python 脚本添加执行权限。

  要完成这个操作，输入以下命令，然后按回车键：

  ```
  chmod +x first_script.py
  ```

  chmod 是一个 Unix 命令，表示**改变访问权限**（change access mode）。+x 表示在访问设置中添加执行权限，而不是添加读权限和写权限，这样 Python 就可以执行脚本中的

代码了。你必须为创建的每个 Python 脚本运行一次 chmod 命令，以使脚本可以执行。只要你在一个文件上运行了 chmod 命令，以后就可以随意运行这个脚本，不用再执行 chmod 命令了。

(4) 运行 Python 脚本。

要完成这个操作，输入以下命令，然后按回车键：

```
./first_script.py
```

你可以看到以下输出在终端窗口中被打印出来，如图 1-6 所示。

```
Output #1: I'm excited to learn Python.
```

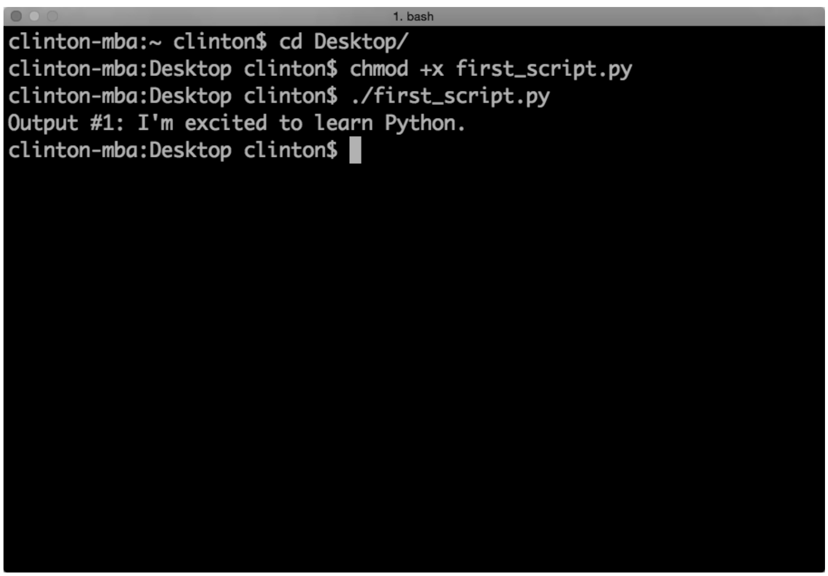


图 1-6：在终端窗口（macOS）中运行 Python 脚本

1.3　与命令行进行交互的几项技巧

下面是与命令行进行交互的几项技巧。

- 使用向上箭头键得到以前的命令

  命令行窗口和终端窗口的一个美妙功能是，你可以通过按向上箭头键找到以前输入的命令。试着在命令行窗口或终端窗口中按一下向上箭头键，找到你以前输入的命令，在 Windows 系统中是 python first_script.py，在 Mac 系统中是 ./first_script.py。

  这个功能非常方便，可以减少每次运行 Python 脚本时必需的输入量，特别是当 Python 脚本的文件名特别长或需要在命令行上提供额外的参数（比如输入文件名或输出文件名）的时候。

- 用 Ctrl+c 停止脚本

  既然你已经学会了运行脚本，那么是时候学习一下如何中断和停止 Python 脚本了。在相当多的情况下，你应该知道如何停止脚本。例如，你可能会写出死循环代码，这样脚本就永远不会停止运行。另外一种情况是，你编写的代码可能需要很长时间才能执行完毕，如果你在代码中包含了 `print` 语句，并由此发现脚本不会生成需要的输出，这时就应该提前终止脚本。

  在脚本开始运行之后，如果想随时中断或停止脚本，可以按 Ctrl+c（Windows）或 Control+c（macOS）。这会停止通过命令开始的进程。你不用太在意这项技术的细节，只要知道**进程**是计算机对一系列命令的处理过程就可以了。你编写了一个**脚本**或**程序**，计算机将它解释成一个进程，如果这个程序非常复杂，就解释成一系列进程，这些进程或者顺序执行，或者并发执行。

- 读懂出错信息并找到解决方案

  这部分的主题是如何处理比较麻烦的脚本，也简单说一下遇到以下问题应该如何解决。当你输入了 `./python first_script.py`，或者试图运行任何一个 Python 脚本的时候，脚本没有正确运行，命令行窗口或终端窗口显示了出错信息。首先不要慌张，先**读懂**出错信息。某些情况下，出错信息中明确指出了代码中出现错误的行，你可以将精力集中在这一行上来纠正错误（你的文本编辑器或 IDE 应该设置成显示行号；如果不自动显示行号，请在菜单中找一下或在网上快速搜索一下，弄清楚如何显示行号）。重要的是要知道出错信息也是编程的一部分，学会编码也包括学会如何有效地调试程序错误。

  更重要的是，因为出错信息是通用的，所以通常很容易学会如何调试程序错误。你很可能不是第一个遇到这种错误并在网上搜索解决方案的人。最好的做法是将整个错误信息（至少是信息的主要部分）复制到搜索引擎（例如：Google 或者 Bing）上，然后在搜索结果中仔细研究其他人是如何调试这种错误的。

  熟悉 Python 内置的异常对象也是非常有帮助的，这样你就可以识别出标准的出错信息并知道如何改正错误。你可以在 Python 标准库页面（http://docs.python.org/3/library/exceptions.html）找到 Python 内置的异常，但是在网上通过出错信息搜索其他人的解决方案也是非常有用的。

- 向 first_script.py 添加更多的代码

  现在，为了更熟练地编写 Python 代码和运行 Python 脚本，试着对 first_script.py 进行编辑，添加更多的代码，然后重新运行脚本。在进行新的练习时，可以把本章提供的代码段添加到脚本中原来代码的下方，保存脚本，然后重新运行。

  举个例子，将下面的两段代码添加到原有的 `print` 语句下面，然后保存脚本并重新运行（请记住，如果你使用的是命令行窗口或终端窗口，在将这些代码添加到 first_script.py 并重新保存脚本之后，可以按向上箭头键，找到你用来运行脚本的命令，不需要将命令重新输入一遍）：

  ```
  # 两个数值相加
  x = 4
  y = 5
  ```

```
z = x + y
print("Output #2: Four plus five equals {0:d}.".format(z))

# 两个列表相加
a = [1, 2, 3, 4]
b = ["first", "second", "third", "fourth"]
c = a + b
print("Output #3: {0}, {1}, {2}".format(a, b, c))
```

以 # 开头的行是注释，用来解释代码，描述代码的用途和目的。

第一个示例展示了将数值赋给变量、变量相加和格式化 print 语句的方法。这里详细说明一下 print 语句中的语法 "{0:d}".format(z)。花括号（{}）是一个占位符，表示这里将要传入 print 语句一个具体的值，这里就是变量 z 的值。0 指向 format() 方法中的第一个参数，在这里，只包含一个参数 z，所以 0 就指向这个值；相反，如果有多个参数，0 就确定地表示传入第一个参数。

冒号（:）用来分隔传入的值和它的格式。d 表示这个值应该被格式化为整数，没有小数部分。在下一节中，你将会学习如何设定小数位数来显示浮点数。

第二个示例展示了创建列表、列表相加和将列表中的值以逗号分隔打印在屏幕上的方法。看一下 print 语句中的语法 "{0}, {1}, {2}".format(a, b, c)，它说明了如何在 print 语句中包含多个值。a 被传给 {0}，b 被传给 {1}，c 被传给 {2}。因为这 3 个值都是列表，不是数值，所以不设置数值格式。本章后面的小节会对这部分内容进行更深入的讨论。

为什么要在打印时使用 .format

Python 并不要求每条 print 语句都必须使用 .format，但是 .format 确实功能强大，可以为你节省很多输入。在上面的示例中，注意 print("Output #3: {0}, {1}, {2}".format(a, b, c)) 的最终结果是**用逗号分隔**的 3 个变量。如果你想在不使用 .format 的情况下得到同样的结果，那么就应该这样写：print("Output #3: ",a,", ",b,", ",c)，但这是一段非常容易出现输入错误的代码。后面还会介绍 .format 的其他用法，但是从现在开始，你就应该熟练掌握它的用法，以便在需要的时候加以使用。

图 1-7 和图 1-8 展示了在 Anaconda Spyder 和 Notepad++ 中添加新代码的界面。

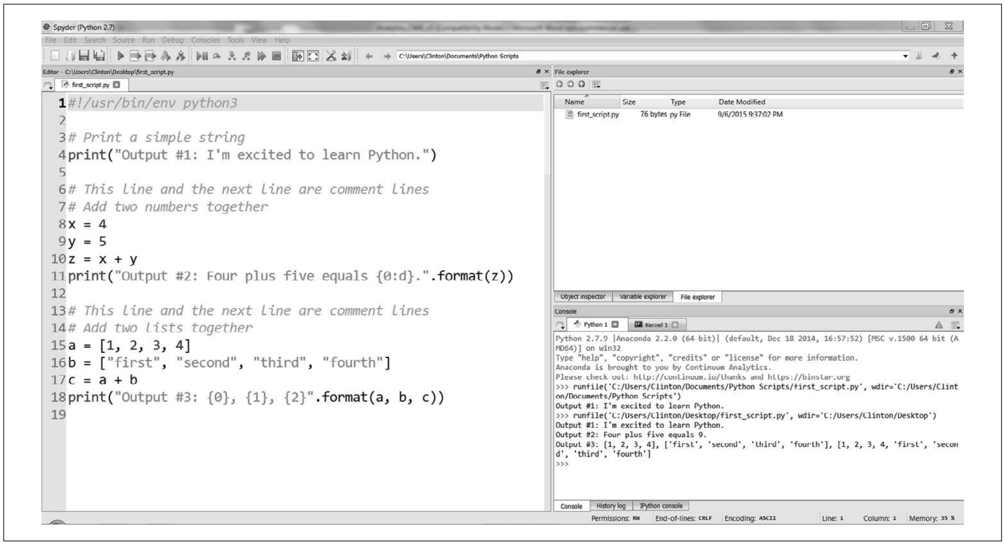


图 1-7：在 Anaconda Spyder 中为 first_script.py 添加新代码

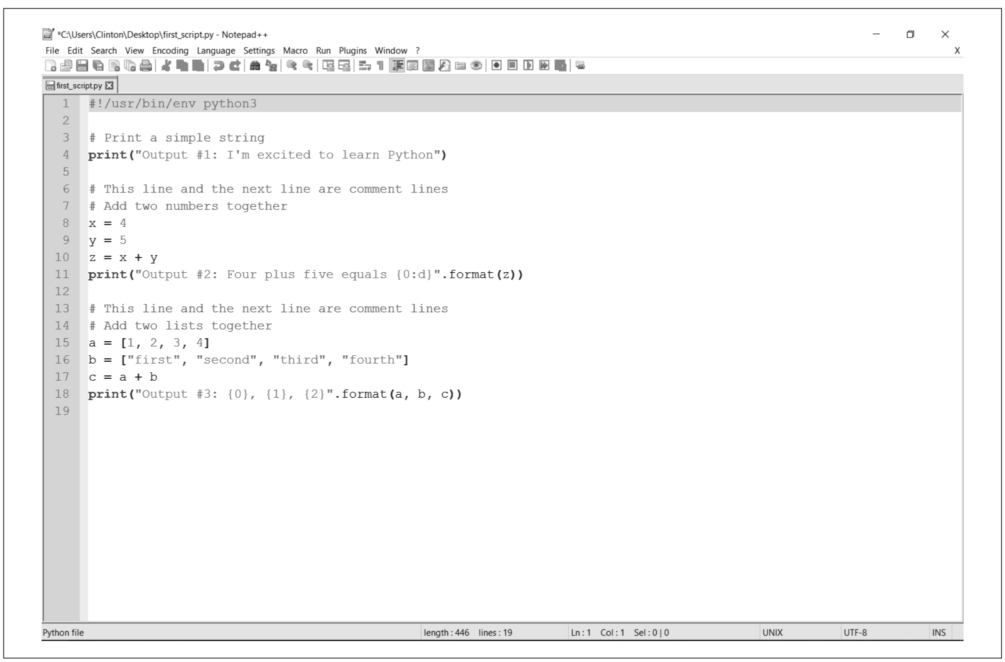


图 1-8：在 Notepad++（Windows）中为 first_script.py 添加新代码

如果你将前面的代码添加到了 first_script.py 中，那么当你保存并且重新运行脚本之后，会看到屏幕中有如下输出（参见图 1-9）：

```
Output #1: I'm excited to learn Python.
```

```
Output #2: Four plus five equals 9.
Output #3: [1, 2, 3, 4], ['first', 'second', 'third', 'fourth'],
[1, 2, 3, 4, 'first', 'second', 'third', 'fourth']
```



图 1-9：在命令行窗口中运行添加了代码之后的 first_script.py

1.4　Python语言基础要素

既然你已经学会了如何创建和运行 Python 脚本，那么以前需要手动实现的业务过程，现在完全可以通过编写 Python 脚本来自动化和规模化地完成。后面几章会详细介绍如何使用 Python 脚本来自动化和规模化地完成任务，但是在进行下一部分内容之前，还需要掌握更多的 Python 语言基础要素。通过掌握更多的基础要素，你会对 Python 有更深入的理解，在后面的章节中就可以综合运用这些知识来完成具体的数据处理任务。首先，本节介绍 Python 中最常用的数据类型，然后讨论使用 `if` 语句和函数来处理数据的方法。在此之后，从实际出发，介绍如何使用 Python 读写文本文件和 CSV 文件。

1.4.1　数值

Python 有好几种内置数值类型。数值类型非常有用，因为很多商业应用需要对数值进行分析和处理。Python 中最主要的 4 种数值类型是整数、浮点数、长整数和复数。这里只介绍整数和浮点数，因为它们在商业应用中是最常用的。你可以把下面处理整数和浮点数的示例添加到 first_script.py 中，放在现有的代码下面，然后重新运行脚本，在屏幕上检查输出。

1. 整数

下面直接看几个带有整数的示例：

```
x = 9
print("Output #4: {0}".format(x))
print("Output #5: {0}".format(3**4))
print("Output #6: {0}".format(int(8.3)/int(2.7)))
```

Output #4 展示了如何将一个整数（数字 9）赋给变量 x，然后将变量 x 打印出来。Output #5 说明了如何得到 3 的 4 次方（等于 81）并将结果打印出来。Output #6 演示了将数值转换成整数并进行除法运算的方法。数值通过内置的 int 函数转换成整数，所以算式变成了 8 除以 2，结果为 4.0。

2. 浮点数

和整数一样，浮点数（即带小数点的数）对很多商业应用来说也是非常重要的。下面是几个带有浮点数的示例：

```
print("Output #7: {0:.3f}".format(8.3/2.7))
y = 2.5*4.8
print("Output #8: {0:.1f}".format(y))
r = 8/float(3)
print("Output #9: {0:.2f}".format(r))
print("Output #10: {0:.4f}".format(8.0/3))
```

Output #7 和 Output #6 非常相似，除了将两个相除的数保留为浮点数，这样算式就是 8.3 除以 2.7，大约等于 3.074。这个示例中 print 语句的语法，"{0:.3f}".format(*floating_point_number*/*floating_point_number*)，说明了如何设置 print 语句中的小数位数。在这个示例中，.3f 设定了打印的输出值应该有 3 位小数。

Output #8 表示用 2.5 乘以 4.8，将结果赋给变量 y，然后将结果打印出来，带有一位小数。这两个浮点数相乘的结果是 12，所以打印出的值是 12.0。Output #9 和 Output #10 表示以两种方式计算 8 除以 3，结果都是一个浮点数，大约等于 2.667。

> **type 函数**
>
> Python 提供一个名为 type 的函数，你可以对所有对象调用这个函数，来获得关于 Python 如何处理这个对象的更多信息。如果你对一个数值变量调用这个函数，它会告诉你这个数值是整数还是浮点数，还会告诉你这个数值是否能当作字符串进行处理。函数的语法非常简单：type(*varible*) 会返回 Python 中的数据类型。此外，因为 Python 是一种“面向对象”的语言，所以你可以对 Python 中所有命名对象调用 type 函数，不仅是变量，还有函数、语句等。如果你的代码出现了意外的错误，调用 type 函数可以帮助你进行错误诊断。

在 Python 中进行数值处理时，需要知道的非常重要的一点是，你可以使用几种标准库模块和内置函数与模块来进行常见的数学计算。你已经使用了两个内置函数来处理数值，分别是 int 和 float。另一个有用的标准模块是 math。

Python 标准模块随着 Python 基本程序一起安装在你的计算机中，但是当你新建一个脚本时，计算机只加载一些非常基本的操作（这就是 Python 启动非常快的一个原因）。要想使用 math 模块中的一些函数，只需在脚本开头 shebang 行的下方添加 from math import [*function name*]。例如，可以将下面的代码行添加到 first_script.py 中，在 shebang 行下面：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
```

如果在 first_script.py 中添加了这一行，你就有 3 个有用的数学函数可以任意使用了。函数 `exp`、`log` 和 `sqrt` 分别表示 `e` 的乘方、自然对数和平方根。下面是使用 `math` 模块中的函数进行计算的几个示例：

```
print("Output #11: {0:.4f}".format(exp(3)))
print("Output #12: {0:.2f}".format(log(4)))
print("Output #13: {0:.1f}".format(sqrt(81)))
```

这 3 种数学函数的计算结果是浮点数，分别约为 20.0855、1.39 和 9.0。

这只是 `math` 模块中一点微小的功能。Python 中还有很多有用的数学函数和模块，用于商业、科学、统计和其他应用，本书还会对此进行更多的讨论。关于数学模块和其他标准模块以及内置函数的更多信 - 息，可以参考 Python 标准库（https://docs.python.org/3/library/index.html）。

1.4.2 字符串

字符串是 Python 中的另一种基本数据类型。它通常是指人类可以阅读的文本，你可以这么理解。但更广泛地说，它是一个字符序列，并且字符只有在组成这个序列时才有意义。很多商业应用中都有字符串类型的数据，比如供应商和客户的名字及地址、评价和反馈数据、事件日志和文档记录。一些对象看上去是整数，但实际上是字符串，比如邮政编码。邮政编码 01111（马萨诸塞州斯普林菲尔德）和整数 1111 是不一样的，你不能对邮政编码做加减乘除，所以最好在代码中将邮政编码作为字符串来处理。这一节将介绍用于字符串管理的一些模块、函数和操作。

字符串可以包含在单引号、双引号、3 个单引号或 3 个双引号之间。下面是字符串的几个示例：

```
print("Output #14: {0:s}".format('I\'m enjoying learning Python.'))

print("Output #15: {0:s}".format("This is a long string. Without the backslash\
it would run off of the page on the right in the text editor and be very\
difficult to read and edit. By using the backslash you can split the long\
string into smaller strings on separate lines so that the whole string is easy\
to view in the text editor."))

print("Output #16: {0:s}".format('''You can use triple single quotes

for multi-line comment strings.'''))

print("Output #17: {0:s}".format("""You can also use triple double quotes

for multi-line comment strings."""))
```

Output #14 和本章开头的示例非常像。它展示了一个包含在单引号之间的简单字符串。这个 `print` 语句的结果是 `"I'm enjoying learning Python."`。请记住，如果用双引号来包含这个字符串的话，就不需要在 `"I'm"` 的单引号前面使用反斜杠了。

Output #15 展示了如何使用反斜杠将一个非常长的字符串分段显示在多个行中，以使它易于理解和编辑。尽管这个字符串分布在脚本的多个行中，它还是一个字符串，并作为一个

完整的字符串被打印出来。在这种将长字符串分成多行的方法中，要注意的是反斜杠必须是每行的**最后**一个字符。如果你意外地按了一下空格键，反斜杠后面就会出现一个看不见的空格，脚本就会抛出一个语法错误，不能正常运行。因此，使用 3 个单引号或 3 个双引号来创建多行字符串更稳妥一些。

Output #16 和 Output #17 展示了如何使用 3 个单引号和 3 个双引号来创建多行字符串。这两个示例的输出如下：

```
Output #16: You can use triple single quotes
for multi-line comment strings.
Output #17: You can also use triple double quotes
for multi-line comment strings.
```

当你使用 3 个单引号或 3 个双引号时，不需要在前面一行的末尾加上反斜杠。还有，请注意一下 Output #15 和 Output #16 以及 Output #17 在打印到屏幕之后的区别。Output #15 使用在行尾添加反斜杠的方式将字符串分成多行，使每行代码更短并且更易于理解，但还是作为一行长文本打印到屏幕上。与之相反，Output #16 和 Output #17 使用 3 个单引号和 3 个双引号创建多行字符串，打印到屏幕上后也是分行的。

和数值类型一样，也有很多标准模块、内置函数和操作符可以用来管理字符串。常用的操作符和函数包括 `+`、`*` 和 `len`。下面就是几个使用这些操作符处理字符串的示例：

```
string1 = "This is a "
string2 = "short string."
sentence = string1 + string2
print("Output #18: {0:s}".format(sentence))
print("Output #19: {0:s} {1:s}{2:s}".format("She is", "very "*4, "beautiful."))
m = len(sentence)
print("Output #20: {0:d}".format(m))
```

Output #18 展示了如何使用 `+` 操作符来将两个字符串相加。这个 `print` 语句的输出结果是 `This is a short string`。`+` 操作符将两个字符串按照原样相加，所以如果你想在结果字符串中留出空格的话，就必须在原字符串中加上空格（例如：在 Output #18 中，要在字母“a”后面加空格）或者在原字符串之间加上空格（例如：在 Output #19 中，要在“very”后面加上空格）。

Output #19 展示了如何使用 `*` 操作符将字符串重复一定的次数。在这个示例中，结果字符串包含了字符串“very”（就是 very 后面跟着一个空格）的 4 个副本。

Output #20 展示了如何使用内置函数 `len` 来确定字符串中字符的数量。函数 `len` 将空格与标点符号也计入字符串长度。所以，在 Output #20 中，字符串 `This is a short string.` 的长度是 23 个字符。

处理字符串的一个常用标准库模块是 `string`。在 `string` 模块中，你可以使用多个函数来有效管理字符串。下面用示例说明了使用这些字符串函数的方法。

1. `split`

下面的两个示例展示了如何使用 `split` 函数来将一个字符串拆分成一个子字符串列表，列表中的子字符串正好可以构成原字符串。（列表是 Python 中的另一种内置数据类型，本章

后面将会讨论。）split 函数可以在括号中使用两个附加**参数**。第一个附加参数表示使用哪个字符进行拆分。第二个附加参数表示进行拆分的次数（例如：进行两次拆分，可以得到 3 个子字符串）：

```
string1 = "My deliverable is due in May"
string1_list1 = string1.split()
string1_list2 = string1.split(" ",2)
print("Output #21: {0}".format(string1_list1))
print("Output #22: FIRST PIECE:{0} SECOND PIECE:{1} THIRD PIECE:{2}"\
.format(string1_list2[0], string1_list2[1], string1_list2[2]))
string2 = "Your,deliverable,is,due,in,June"
string2_list = string2.split(',')
print("Output #23: {0}".format(string2_list))
print("Output #24: {0} {1} {2}".format(string2_list[1], string2_list[5],\
string2_list[-1]))
```

在 Output #21 中，括号中没有附加参数，所以 split 函数使用空格字符（默认值）对字符串进行拆分。因为这个字符串中有 5 个空格，所以字符串被拆分成具有 6 个子字符串的列表。新生成的列表为 ['My', 'deliverable', 'is', 'due', 'in', 'May']。

Output #22 明确地在 split 函数中包含了两个附加参数。第一个附加参数是 " "，说明想用空格来拆分字符串。第二个附加参数是 2，说明只想使用前两个空格进行拆分。因为设定了拆分两次，所以会生成一个带有 3 个元素的列表。第二个附加参数会在你解析数据的时候派上用场。举例来说，你可能会解析一个日志文件，文件中包含时间戳、错误代码和由空格分隔的错误信息。在这种情况下，你应该使用前两个空格进行拆分，解析出时间戳和错误代码，但是不使用剩下的空格进行拆分，以便完整无缺地保留错误信息。

在 Output #23 和 Output #24 中，括号中的附加参数是个逗号。在这种情况下，split 函数在出现逗号的位置拆分字符串。结果列表为 ['Your', 'deliverable', 'is', 'due', 'in', 'June']。

2. join

下面的示例展示了如何使用 join 函数将列表中的子字符串组合成一个字符串。join 函数将一个参数放在 join 前面，表示使用这个字符（或字符串）在子字符串之间进行组合：

```
print("Output #25: {0}".format(','.join(string2_list)))
```

在这个示例中，附加参数为一个逗号，位于圆括号中。所以 join 函数将子字符串组合成一个字符串，子字符串之间为逗号。因为列表中有 6 个子字符串，所以子字符串被组合成一个字符串，子字符串之间有 5 个逗号。新生成的字符串是 Your,deliverable,is,due,in,June。

3. strip

下面两组示例展示了如何使用 strip、lstrip 和 rstrip 函数从字符串两端删除不想要的字符。这 3 个函数都可以在括号中使用一个附加参数来设定要从字符串两端删除的字符（或字符串）。

第一组示例展示了如何使用 lstrip、rstrip 和 strip 函数分别从字符串的左侧、右侧和两侧删除空格、制表符和换行符：

```
string3 = " Remove unwanted characters    from this string.\t\t    \n"
print("Output #26: string3: {0:s}".format(string3))
string3_lstrip = string3.lstrip()
print("Output #27: lstrip: {0:s}".format(string3_lstrip))
string3_rstrip = string3.rstrip()
print("Output #28: rstrip: {0:s}".format(string3_rstrip))
string3_strip = string3.strip()
print("Output #29: strip: {0:s}".format(string3_strip))
```

string3 的左侧含有几个空格。此外，右侧包含制表符（\t）、几个空格和换行符（\n）。如果以前你没有见过 \t 和 \n，那么现在知道了这是计算机中表示制表符和换行符的方法。

在 Output #26 中，你会看到句子前面有空白字符；在句子下面有一个空行，因为有换行符；在句子后面你看不到制表符和空格，但它们确实在那儿。Output #27、Output #28 和 Output #29 分别展示了从字符串左侧、右侧和两侧删除空格、制表符和换行符的方法。{0:s} 中的 s 表示传入 print 语句的值应该格式化为一个字符串。

第二组示例展示了从字符串两端删除其他字符的方法，将这些字符作为 strip 函数的附加参数即可。

```
string4 = "$$Here's another string that has unwanted characters.__---++"
print("Output #30: {0:s}".format(string4))
string4 = "$$The unwanted characters have been removed.__---++"
string4_strip = string4.strip('$_-+')
print("Output #31: {0:s}".format(string4_strip))
```

在这组示例中，美元符号（$）、下划线（_）、短划线（-）和加号（+）需要从字符串两端删除。通过将这些字符作为附加参数，可以通知程序从字符串两端删除它们。在 Output #31 中，结果字符串为 The unwanted characters have been removed.。

4. replace

下面两个示例展示了如何使用 replace 函数将字符串中的一个或一组字符替换为另一个或另一组字符。这个函数在括号中使用两个附加参数，第一个参数是要在字符串中查找替换的字符或一组字符，第二个参数是要用来替换掉第一个参数的字符或一组字符：

```
string5 = "Let's replace the spaces in this sentence with other characters."
string5_replace = string5.replace(" ", "!@!")
print("Output #32 (with !@!): {0:s}".format(string5_replace))
string5_replace = string5.replace(" ", ",")
print("Output #33 (with commas): {0:s}".format(string5_replace))
```

Output #32 展示了如何使用 replace 函数将字符串中的空格替换为 !@!。结果字符串为 Let's!@!replace!@!the!@!spaces !@!in!@!this!@!sentence!@!with!@!other!@!characters.。

Output #33 展示了如何使用逗号替换字符串中的空格。结果字符串为 Let's,replace,the,spaces,in,this,sentence,with,other,characters.。

5. lower、upper、capitalize

最后 3 个示例展示了如何使用 lower、upper 和 capitalize 函数。lower 和 upper 函数分别用来将字符串中的字母转换为小写和大写。capitalize 函数对字符串中的第一个字母应用

upper 函数，对其余的字母应用 lower 函数：

```
string6 = "Here's WHAT Happens WHEN You Use lower."
print("Output #34: {0:s}".format(string6.lower()))
string7 = "Here's what Happens when You Use UPPER."
print("Output #35: {0:s}".format(string7.upper()))
string5 = "here's WHAT Happens WHEN you use Capitalize."
print("Output #36: {0:s}".format(string5.capitalize()))
string5_list = string5.split()
print("Output #37 (on each word):")
for word in string5_list:
    print("{0:s}".format(word.capitalize()))
```

Output #34 和 Output #35 是对 lower 和 upper 函数最直接的应用。在字符串上应用了这些函数之后，string6 中的所有字母都是小写，string7 中的所有字母都是大写。

Output #36 和 Output #37 演示了 capitalize 函数。Output #36 说明了 capitalize 函数对字符串中的第一个字母应用 upper 函数，对其余的字母应用 lower 函数。Output #37 将 capitalize 函数放在一个 for 循环中。for 循环是一个控制流结构，将在后面详细讨论，现在不妨先看一下。

代码 for word in string5_list 的意义是这样的："对于 string5_list 这个列表中的每个元素，需要做点事情。"下面的代码 print word.capitalize() 说明了对于列表中的每个元素要做的事情。这两行代码放在一起的意义就是："对于 string5_list 这个列表中的每个元素，先应用 capitalize 函数，然后再打印出来。"代码执行的结果就是列表中的每个单词首字母大写，其余字母小写。

Python 中还有更多管理字符串的模块和函数。和内置函数 math 一样，你可以参考 Python 标准库（https://docs.python.org/3/library/index.html）来获取更多信息。

1.4.3 正则表达式与模式匹配

很多商业分析都依赖模式匹配，也称为**正则表达式**（regular expression）。举例来说，你可能需要分析一下来自某个州（比如马里兰州）的所有订单。在这种情况下，你需要识别的模式就是 Maryland 这个单词。同样，你还可能需要分析一下来自某个供应商（比如 StaplesRUs）的商品质量，那么你要识别的模式就是 StaplesRUs。

Python 包含了 re 模块，它提供了在文本中搜索特定模式（也就是正则表达式）的强大功能。要在脚本中使用 re 模块提供的功能，需要在脚本上方加入 import re 这行代码，放在上一个 import 语句之后。现在 first_script.py 的上方应该是这样的：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
import re
```

通过导入 re 模块，你可以使用一大波函数和元字符来创建和搜索任意复杂的模式。**元字符**（metacharacter）是正则表达式中具有特殊意义的字符。每个元字符都有特殊意义，它们使正则表达式能够匹配特定的字符串。常用的元字符包括 |、()、[]、.、*、+、?、^、$ 和 (?P<*name*>)。如果你在正则表达式中见到这些字符，要知道程序不是要搜索这些字

符本身，而是要搜索它们描述的东西。你可以在 Python 标准库中的“Regular Expression Operations”一节中获得关于元字符的更多信息（https://docs.python.org/3/library/re.html）。

re 模块中还包括很多有用的函数，用于创建和搜索特定的模式（本节要介绍的函数包括 re.compile、re.search、re.sub、re.ignorecase 和 re.I）。来看一下示例代码：

```
# 计算字符串中模式出现的次数
string = "The quick brown fox jumps over the lazy dog."
string_list = string.split()
pattern = re.compile(r"The", re.I)
count = 0
for word in string_list:
    if pattern.search(word):
        count += 1
print("Output #38: {0:d}".format(count))
```

第一行将字符串变量 string 赋值为 The quick brown fox jumps over the lazy dog.。下一行将字符串拆分列表，列表中的每个元素都是一个单词。

再下一行使用 re.compile 和 re.I 函数以及用 r 表示的原始字符串，创建一个名为 pattern 的正则表达式。re.compile 函数将文本形式的模式编译成为编译后的正则表达式。正则表达式不是必须编译的，但是编译是个好习惯，因为这样可以显著地提高程序运行速度。re.I 函数确保模式是不区分大小写的，能同时在字符串中匹配“The”和“the”。原始字符串标志 r 可以确保 Python 不处理字符串中的转义字符，比如 \、\t 或 \n。这样在进行模式匹配时，字符串中的转义字符和正则表达式中的元字符就不会有意外的冲突。在上面的例子中，字符串中没有转义字符，所以 r 不是必需的，但是在正则表达式中使用原始字符串标志是一个好习惯。接下来的一行代码创建了一个变量 count 来保存字符串中模式出现的次数，初始值设为 0。

再下一行是个 for 循环语句，在列表变量 string_list 的各个元素之间进行迭代。它取出的第一个元素是“The”这个单词，取出的第二个元素是“quick”这个单词，以此类推，直到取出列表中所有的单词。接下来的一行使用 re.search 函数将列表中的每个单词与正则表达式进行比较。如果这个单词与正则表达式相匹配，函数就返回 True，否则就返回 None 或 False。所以 if 语句的意义就是：如果单词与正则表达式匹配，那么 count 的值就加 1。

最后，print 语句打印出正则表达式在字符串中找到模式“The”（不区分大小写）的次数，在本例中，找到了两次。

太可怕了！

正则表达式在进行搜索时的确功能强大，但是非常难以读懂（它曾被称为“只用来写的语言”），所以当你第一次没有看懂的时候，不用太在意，连专家都不一定能看懂！

当你逐渐熟悉了正则表达式后，使用它们得到你想要的结果就简单了。要想愉快地学习正则表达式，你可以参考一下 Google 研发总监 Peter Norvig 的工作，他创建了一个正则表达式，这个表达式可以匹配美国总统的名字，并且可以筛掉失败的总统竞选人，网址为 https://www.oreilly.com/learning/regex-golf-with-peter-norvig。

再看另一个示例：

```
# 在字符串中每次找到模式时将其打印出来
string = "The quick brown fox jumps over the lazy dog."
string_list = string.split()
pattern = re.compile(r"(?P<match_word>The)", re.I)</match_word>
print("Output #39:")
for word in string_list:
    if pattern.search(word):
        print("{:s}".format(pattern.search(word).group('match_word')
```

第二个示例与第一个示例的区别在于，这个示例是想在屏幕上打印出每个匹配的字符串，而不是匹配的次数。要想得到匹配的字符串，将它们打印到屏幕上或存储在文件中，需要使用 (?P<*name*>) 元字符和 group 函数。这个示例中的多数代码和前一个示例中讨论过的代码是一样的，所以这里重点解释新的部分。

第一个新代码片段是 (?P<*name*>)，这是一个出现在 re.compile 函数中的元字符。这个元字符使匹配的字符串可以在后面的程序中通过组名符号 <*name*> 来引用。在这个示例中，这个组被称为 <match_word>。

最后一个新代码片段出现在 if 语句中。这个代码片段的意义是："如果结果为 True（也就是说，如果单词与模式匹配），那么就在 search 函数返回的数据结构中找出 match_word 组中的值，并把这些值打印在屏幕上。"

下面是最后一个示例：

```
# 使用字母"a"替换字符串中的单词"the"
string = "The quick brown fox jumps over the lazy dog."
string_to_find = r"The"
pattern = re.compile(string_to_find, re.I)
print("Output #40: {:s}".format(pattern.sub("a", string)
```

最后一个示例展示了如何使用 re.sub 函数来在文本中用一种模式替换另一种模式。再说一遍，这个示例中的多数代码和前两个示例中讨论过的代码是一样的，所以这里重点解释新的部分。

第一个新代码片段将正则表达式赋给变量 pattern，于是这个变量可以被传入到 re.compile 函数中。解释一下，在调用 re.compile 函数之前，将正则表达式赋给变量不是必需的；但是，如果你的正则表达式特别长而且复杂的话，将它赋给一个变量然后将变量传给 re.compile 函数这种做法可以使你的代码更利于理解。

最后一个新代码片段在最后一行。这个代码片段使用 re.sub 函数以不区分大小写的方式在变量 string 中寻找模式，然后将发现的每个模式替换成字母 a。这次替换的最终结果是 a quick brown fox jumps over a lazy dog.。

更多关于正则表达式函数的信息可以在 Python 标准库（https://docs.python.org/3/library/index.html）中找到，也可以参考 Michael Fitzgerald 的著作《学习正则表达式》[1]。

注 1：此书已由人民邮电出版社出版。——编者注

1.4.4 日期

日期在大多数商业应用中都是必不可少的。你需要知道一个事件在何时发生，距离这件事发生还有多少时间，或者几个事件之间的时间间隔。因为日期是很多应用的核心，也因为日期是一种非常不寻常的数据，在处理时经常要乘以 60 或 24，也有“差不多 30 分钟”和“几乎是 365 天和一个季度”这样的说法，所以在 Python 中对日期有特殊的处理方式。

Python 中包含了 datetime 模块，它提供了非常强大的功能来处理日期和时间。要想在脚本中使用 datetime 模块提供的功能，需要在脚本上方加入 from datetime import date, time, datetime, timedelta，放在之前的 import 语句下面。现在 first_script.py 的上方应该是这样的：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
import re
from datetime import date, time, datetime, timedelta
```

导入 datetime 模块之后，就有各式各样的日期时间对象和函数供你随意使用了。常用的对象和函数包括 today、year、month、day、timedelta、strftime 和 strptime。这些函数可以捕获具体的日期数据（例如：年、月、日）、进行日期和时间的加减运算、创建特定形式的日期字符串以及根据日期字符串创建 datetime **对象**。下面是使用这些 datetime 对象和函数的几个示例。第一组示例演示了 date 对象和 datetime 对象之间的区别：

```
# 打印出今天的日期形式,以及年、月、日
today = date.today()
print("Output #41: today: {0!s}".format(today))
print("Output #42: {0!s}".format(today.year))
print("Output #43: {0!s}".format(today.month))
print("Output #44: {0!s}".format(today.day))
current_datetime = datetime.today()
print("Output #45: {0!s}".format(current_datetime))
```

通过使用 date.today()，你可以创建一个 date 对象，其中包含了年、月、日，但不包含时间元素，比如时、分、秒。相反，通过 datetime.today() 创建的对象则包含时间元素。{0!s} 中的 !s 表示传入到 print 语句中的值应该格式化为字符串，尽管它是个数值型数据。最后，你可以使用 year、month 和 day 来捕获具体的日期元素。

下一个示例演示了如何使用 timedelta 函数来对 date 对象进行时间的加减操作：

```
# 使用timedelta计算一个新日期
one_day = timedelta(days=-1)
yesterday = today + one_day
print("Output #46: yesterday: {0!s}".format(yesterday))
eight_hours = timedelta(hours=-8)
print("Output #47: {0!s} {1!s}".format(eight_hours.days, eight_hours.seconds))
```

在这个示例中，使用 timedelta 函数从今天减去了 1 天。当然，还可以在括号中使用 days=10、hours=-8 或者 weeks=2 来创建变量，分别表示未来 10 天、以前 8 个小时或者未来 2 个星期。

在使用 timedelta 时需要注意的一点是，它将括号中的时间差以天、秒和毫秒的形式存储，然后将数值规范化后得到一个唯一的值。这说明分钟、小时和星期会被分别转换成 60 秒、3600 秒和 7 天，然后规范化，就是生成天、秒和毫秒“列”（类似于小学数学中的个位、十位 等等）。举例来说，hours=-8 的输出是 (-1 days, 57,600 seconds)，不是更简单的 (-28,800 seconds)。是这样计算的：86 400 秒（3600 秒每小时 *24 小时每天）−28 800 秒（3600 秒每小时 *8 小时）= 57 600 秒。正如你所见，对负值的规范化乍看上去很令人吃惊，特别是在进行取整和舍入时。

第三个示例展示了如何从一个 date 对象中减去另一个。相减的结果是个 datetime 对象，将所得的差以天、小时、分钟和秒来显示。例如，在这个示例中结果是“1 day, 0:00:00”：

```
# 计算出两个日期之间的天数
date_diff = today - yesterday
print("Output #48: {0!s}".format(date_diff))
print("Output #49: {0!s}".format(str(date_diff).split()[0]))
```

在某些情况下，你可能只需要结果中的数值部分。举例来说，在这个示例中你只需要数值 1。从结果中得到这个数值的一种方法是使用前面已经讨论过的字符串函数。str 函数可以将结果转换成字符串；split 函数可以使用空白字符将字符串拆分，并使每个子字符串成为列表的一个元素；[0] 表示“取出列表中的第一个元素”，在本例中就是数值 1。在 1.4.5 节中还会看到 [0] 这种语法，因为它是列表索引，可以用来从列表中取出特定的元素。

第四组示例展示了如何使用 strftime 函数根据一个 date 对象来创建具有特定格式的字符串：

```
# 根据一个日期对象创建具有特定格式的字符串
print("Output #50: {:s}".format(today.strftime('%m/%d/%Y')))
print("Output #51: {:s}".format(today.strftime('%b %d, %Y')))
print("Output #52: {:s}".format(today.strftime('%Y-%m-%d')))
print("Output #53: {:s}".format(today.strftime('%B %d, %Y')))
```

在我写这一章的时候，当天的 4 种打印形式如下：

```
01/28/2016
Jan 28, 2016
2016-01-28
January 28, 2016
```

这 4 个示例说明了如何使用格式符来创建不同格式的日期字符串，格式符包括 %Y、%B、%b、%m 和 %d。你可以在 Python 标准库（https://docs.python.org/3/library/datetime.html）的“datetime—Basic date and time types”一节中找到 datetime 模块使用的其他格式符。

```
# 根据一个表示日期的字符串
# 创建一个带有特殊格式的datetime对象
date1 = today.strftime('%m/%d/%Y')
date2 = today.strftime('%b %d, %Y')
date3 = today.strftime('%Y-%m-%d')
date4 = today.strftime('%B %d, %Y')

# 基于4个具有不同日期格式的字符串
# 创建2个datetime对象和2个date对象
print("Output #54: {!s}".format(datetime.strptime(date1, '%m/%d/%Y')))
print("Output #55: {!s}".format(datetime.strptime(date2, '%b %d, %Y')))
```

```
# 仅显示日期部分
print("Output #56: {!s}".format(datetime.date(datetime.strptime\
(date3, '%Y-%m-%d'))))
print("Output #57: {!s}".format(datetime.date(datetime.strptime\
(date4, '%B %d, %Y'))))
```

第五组示例展示了如何使用 strptime 函数根据具有特定形式的日期字符串来创建 datetime 对象。在这个示例中，date1、date2、date3 和 date4 是字符串变量，以不同的形式表示今天。前两个 print 语句展示了将前两个字符串变量 date1 和 date2 转换成 datetime 对象的结果。要使它们正确工作，strptime 函数中使用的形式需要和传入函数的字符串变量的形式相匹配。这两个 print 语句的输出结果是个 datetime 对象，2014-01-28 00:00:00。

有些时候，你可能只对 datetime 对象中的日期部分感兴趣。在这种情况下，你可以使用嵌套的函数（在最后两个 print 语句中，是 date 和 strptime）将日期字符串变量转换为 datetime 对象，然后仅返回 datetime 对象的日期部分。这些 print 语句的结果是 2014-01-28。当然，你不需要立刻打印出这个值。你可以将这个日期赋给一个新变量，然后使用这个变量进行计算，洞悉随着时间产生的业务数据。

1.4.5 列表

很多商业分析中都使用列表。你会维护各种客户列表、产品列表、资产列表、销售量列表，等等。但是 Python 中的列表（对象的可排序集合）更加灵活！上面那些列表中包含的都是相似的对象（例如：包含客户姓名的字符串或代表销售量的浮点数），但是 Python 中的列表可不止这么简单。它可以包含数值、字符串、其他列表、元组和字典（本章稍后介绍）的任意组合。因为列表在商业应用中使用广泛、灵活性高、作用突出，所以掌握如何在 Python 中操作列表是极其重要的。

如你所愿，Python 提供了很多有用的函数和操作符来管理列表。以下演示了最常用的和最有效的列表函数和操作符的使用方法。

1. 创建列表

```
# 使用方括号创建一个列表
# 用len()计算列表中元素的数量
# 用max()和min()找出最大值和最小值
# 用count()计算出列表中某个值出现的次数
a_list = [1, 2, 3]
print("Output #58: {}".format(a_list))
print("Output #59: a_list has {} elements.".format(len(a_list)))
print("Output #60: the maximum value in a_list is {}.".format(max(a_list)))
print("Output #61: the minimum value in a_list is {}.".format(min(a_list)))
another_list = ['printer', 5, ['star', 'circle', 9]]
print("Output #62: {}".format(another_list))
print("Output #63: another_list also has {} elements.".format\
(len(another_list)))
print("Output #64: 5 is in another_list {} time.".format(another_list.count(5)))
```

这个示例展示了如何创建两个简单列表，a_list 和 another_list。将元素放在方括号之间就可以创建列表。a_list 包含数值 1、2 和 3。another_list 包含一个字符串 printer、一

个数值 5 以及一个包含了两个字符串和一个数值的列表。

这个示例还展示了如何使用 4 种列表函数：len、min、max 和 count。len 返回列表中元素的个数。min 和 max 分别返回列表中的最小值和最大值。count 返回列表中某个元素出现的次数。

2. 索引值

```
# 使用索引值访问列表中的特定元素
# [0]是第1个元素,[-1]是最后一个元素
print("Output #65: {}".format(a_list[0]))
print("Output #66: {}".format(a_list[1]))
print("Output #67: {}".format(a_list[2]))
print("Output #68: {}".format(a_list[-1]))
print("Output #69: {}".format(a_list[-2]))
print("Output #70: {}".format(a_list[-3]))
print("Output #71: {}".format(another_list[2]))
print("Output #72: {}".format(another_list[-1]))
```

这个示例说明了如何使用索引值来引用列表中的特定元素。列表索引值从 0 开始，所以你可以通过在列表名称后面的方括号中放入 0 来引用列表中的第一个元素。示例中的第一个 print 语句 print a_list[0] 打印出 a_list 中的第一个元素，即数值 1。a_list[1] 引用的是列表中的第二个元素，a_list[2] 引用的是列表中的第三个元素，以此类推直到列表结束。

这个示例还说明了你可以使用负索引值从列表尾部引用列表元素。列表尾部的索引值从 −1 开始，所以你可以通过在列表名称后面的方括号中放入 −1 来引用列表中的最后一个元素。第四个 print 语句 print a_list[-1] 打印出 a_list 中的最后一个元素，即数值 3。a_list[-2] 打印出列表中倒数第二个元素，a_list[-3] 打印出列表中倒数第三个元素，以此类推直到列表开头。

3. 列表切片

```
# 使用列表切片访问列表元素的一个子集
# 从开头开始切片,可以省略第1个索引值
# 一直切片到末尾,可以省略第2个索引值
print("Output #73: {}".format(a_list[0:2]))
print("Output #74: {}".format(another_list[:2]))
print("Output #75: {}".format(a_list[1:3]))
print("Output #76: {}".format(another_list[1:]
```

这个示例展示了如何使用列表切片引用列表元素的一个子集。在列表名称后面的方括号中放入由冒号隔开的两个索引，就可以创建一个列表切片。列表切片引用的是列表中从第一个索引值到第二个索引值的前一个元素。举例来说，第一个 print 语句 print a_list[0:2] 的意义就是：“打印出 a_list 中索引值为 0 和 1 的元素”。这个 print 语句打印出 [1, 2]，因为这是列表中的前两个元素。

这个示例还说明，如果从列表开头开始切片，就可以省略第一个索引值，如果一直切片到列表末尾，就可以省略第二个索引值。举例来说，最后一个 print 语句 print another_list[1:] 的意义就是：“从列表中第二个元素开始，打印出 another_list 中其余所有的元素。”这个 print 语句打印出 [5, ['star', 'circle', 9]]，因为这是列表中最后两个元素。

4. 列表复制

```
# 使用[:]复制一个列表
a_new_list = a_list[:]
print("Output #77: {}".format(a_new_list))
```

这个示例展示了如何复制一个列表。如果你需要对列表进行某种操作，比如添加或删除元素，或对列表进行排序，但你还希望原始列表保持不变，这时这个功能就非常重要了。要复制一个列表，在列表名称后面的方括号中放入一个冒号，然后将其赋给一个新的变量即可。在这个示例中，a_new_list 是 a_list 的一个完美复制，所以你可以对 a_new_list 添加或删除元素，也可以对 a_new_list 进行排序，而不会影响 a_list。

5. 列表连接

```
# 使用+将两个或更多个列表连接起来
a_longer_list = a_list + another_list
print("Output #78: {}".format(a_longer_list))
```

这个示例展示了如何将两个或更多个列表连接在一起。当你必须分别引用多个具有相似信息的列表，但希望将它们组合起来进行分析的时候，这个功能就非常重要了。举例来说，由于数据存储方式的原因，你可能需要生成一个销售量列表，其中包括来自于一个数据源和另一个数据源的销售量列表。要将两个销售量列表连接在一起进行分析，可以将两个列表名称用 + 操作符相加，然后赋给一个新变量。在这个示例中，a_long_list 包含 a_list 和 another_list 中的所有元素，将它们连接在一起形成一个更长的列表。

6. 使用in和not in

```
# 使用in和not in来检查列表中是否有特定元素
a = 2 in a_list
print("Output #79: {}".format(a))
if 2 in a_list:
    print("Output #80: 2 is in {}.".format(a_list))
b = 6 not in a_list
print("Output #81: {}".format(b))
if 6 not in a_list:
    print("Output #82: 6 is not in {}.".format(a_list))
```

这个示例展示了如何使用 in 和 not in 来检查列表中是否存在某个特定元素。这些表达式的结果是 True 或 False，取决于表达式为真还是假。这个功能在商业应用中非常重要，因为你可以使用它在程序中添加有意义的业务逻辑。举例来说，它们经常应用于 if 语句，比如“如果 SupplierY 在 SupplierList 中，那么做些什么事，否则做些其他事。”本章后面的内容中将会介绍更多 if 语句和其他控制流表达式的示例。

7. 追加、删除和弹出元素

```
# 使用append()向列表末尾追加一个新元素
# 使用remove()从列表中删除一个特定元素
# 使用pop()从列表末尾删除一个元素
a_list.append(4)
a_list.append(5)
a_list.append(6)
print("Output #83: {}".format(a_list))
a_list.remove(5)
```

```
print("Output #84: {}".format(a_list))
a_list.pop()
a_list.pop()
print("Output #85: {}".format(a_list))
```

这个示例展示了向列表中添加元素和从列表中删除元素的方法。append 方法将一个新元素追加到列表末尾。你可以使用该方法按照具体业务规则创建列表。举例来说，要建立一个关于 CustomerX 的采购量的列表，可以先创建一个名为 CustomerX 的空列表，然后在记录所有客户采购量的主列表中扫描，如果在主列表中发现了 CustomerX，就可以将这个采购量数据追加到列表 CustomerX 中。

remove 方法可以删除列表中的任意元素。你可以使用该方法删除列表中的错误元素和输入错误，还可以按照具体业务规则删除列表中的元素。在这个示例中，remove 方法从 a_list 中删除了数值 5。

pop 方法删除列表中的最后一个元素。和 remove 方法相似，你可以使用 pop 方法删除列表末尾的错误元素和输入错误，还可以按照具体的业务规则从列表末尾删除元素。在这个示例中，对 pop 方法的两次调用从 a_list 中分别删除了数值 6 和数值 4。

8. 列表反转

```
# 使用reverse()原地反转一个列表会修改原列表
# 要想反转列表同时又不修改原列表,可以先复制列表
a_list.reverse()
print("Output #86: {}".format(a_list))
a_list.reverse()
print("Output #87: {}".format(a_list))
```

这个示例展示了使用 reverse 函数以 in-place 方式对列表进行反转的方法（原地反转）。“in-place”表示反转操作将原列表修改为顺序颠倒的新列表。举例来说，示例第一次调用 reverse 函数将 a_list 改变为 [3, 2, 1]。第二次调用 reverse 函数则将 a_list 恢复到初始顺序。要想使用列表的反转形式而不修改原列表，可以先复制列表，然后对列表副本进行 reverse 操作。

9. 列表排序

```
# 使用sort()对列表进行原地排序会修改原列表
# 要想对列表进行排序同时又不修改原列表,可以先复制列表
unordered_list = [3, 5, 1, 7, 2, 8, 4, 9, 0, 6]
print("Output #88: {}".format(unordered_list))
list_copy = unordered_list[:]
list_copy.sort()
print("Output #89: {}".format(list_copy))
print("Output #90: {}".format(unordered_list))
```

这个示例展示了使用 sort 函数以 in-place 方式对列表进行排序的方法。和 reverse 函数一样，这种原地排序将原列表修改为排好顺序的新列表。要想使用排好顺序的列表而不修改原列表，可以先复制列表，然后对列表副本进行 sort 操作。

10. sorted排序函数

```
# 使用sorted()对一个列表集合按照列表中某个位置的元素进行排序
```

```
my_lists = [[1,2,3,4], [4,3,2,1], [2,4,1,3]]
my_lists_sorted_by_index_3 = sorted(my_lists, key=lambda index_value:\
index_value[3])
print("Output #91: {}".format(my_lists_sorted_by_index_3))
```

这个示例展示了如何使用 sorted 函数以及关键字函数，对一个列表集合按照每个列表中特定索引位置的值进行排序。关键字函数设置用于列表排序的关键字。在这个示例中，关键字是一个 lambda 函数，表示使用索引位置为 3 的值（也就是列表中的第四个元素）对列表进行排序。（后续章节将会对 lambda 函数做更多讨论。）使用每个列表中的第四个元素作为排序关键字，应用 sorted 函数之后，第二个列表 [4, 3, 2, 1] 成为了第一个列表，第三个列表 [2, 4, 1, 3] 成为了第二个列表，第一个列表 [1, 2, 3, 4] 成为了第三个列表。另外，你应该知道 sorted 函数的排序与 sort 函数的 in-place 原地排序方式不同，sort 函数改变了原列表的元素顺序，sorted 函数则返回一个新的排好序的列表，并不改变原列表的元素顺序。

下一个排序示例使用 operator 标准模块，这个模块提供的功能可以使用多个关键字对列表、元组和字典进行排序。为了在脚本中使用 operator 模块中的 itemgetter 函数，在脚本上方添加 from operator import itemgetter：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
import re
from datetime import date, time, datetime, timedelta
from operator import itemgetter
```

导入 operator 模块中的 itemgetter 函数后，你可以使用每个列表中多个索引位置的值对列表集合进行排序：

```
# 使用itemgetter()对一个列表集合按照两个索引位置来排序
my_lists = [[123,2,2,444], [22,6,6,444], [354,4,4,678], [236,5,5,678], \
[578,1,1,290], [461,1,1,290]]
my_lists_sorted_by_index_3_and_0 = sorted(my_lists, key=itemgetter(3,0))
print("Output #92: {}".format(my_lists_sorted_by_index_3_and_0))
```

这个示例展示了如何使用 sorted() 函数和 itemgetter 函数按照每个列表中多个索引位置的值对列表集合进行排序。关键字函数设置用于列表排序的关键字。在这个示例中，关键字是包含两个索引值（3 和 0）的 itemgetter 函数。这个语句的意义是："先按照索引位置 3 中的值对列表进行排序，然后，在这个排序基础上，按照索引位置 0 中的值对列表进一步排序。"

这种通过多个元素对列表和其他数据容器进行排序的方法非常重要，因为你经常需要按照多个值对数据进行排序。例如，如果要处理每天销售交易数据，你需要先按照日期再按照每天的交易量大小进行排序。或者，在处理供应商数据时，你会先按照供应商姓名再按照每个供应商的供货发票日期来排序。sorted 函数和 itemgetter 函数可以实现这样的功能。

如果想获得列表函数的更多信息，可以参考 Python 标准库（https://docs.python.org/3/library/index.html）。

1.4.6　元组

元组除了不能被修改之外，其余特点与列表非常相似。正因为元组不能被修改，所以没有元组修改函数。你可能会感到奇怪，为什么要设计这两种如此相似的数据结构。这是因为元组具有可修改的列表无法实现的重要作用，例如作为字典键值。元组不如列表使用广泛，所以这里只是简略地介绍一下。

1. 创建元组

```
# 使用圆括号创建元组
my_tuple = ('x', 'y', 'z')
print("Output #93: {}".format(my_tuple))
print("Output #94: my_tuple has {} elements".format(len(my_tuple)))
print("Output #95: {}".format(my_tuple[1]))
longer_tuple = my_tuple + my_tuple
print("Output #96: {}".format(longer_tuple))
```

这个示例展示了创建元组的方法。将元素放在括号中间，就可以创建一个元组。此示例还说明，前面讨论过的很多应用于列表的函数和操作符，也同样适用于元组。例如，`len` 函数返回元组中元素的个数，元组索引和元组切片可以引用元组中特定的元素，+ 操作符可以连接多个元组。

2. 元组解包

```
# 使用赋值操作符左侧的变量对元组进行解包
one, two, three = my_tuple
print("Output #97: {0} {1} {2}".format(one, two, three))
var1 = 'red'
var2 = 'robin'
print("Output #98: {} {}".format(var1, var2))

# 在变量之间交换彼此的值
var1, var2 = var2, var1
print("Output #99: {} {}".format(var1, var2))
```

这个示例展示了元组的一个很有意思的操作——**解包**。可以将元组中的元素解包成为变量，在赋值操作符的左侧放上相应的变量就可以了。在这个示例中，字符串 `x`、`y` 和 `z` 被解包成为变量 `one`、`two` 和 `three` 的值。这个功能可以用来在变量之间交换变量值。在示例的最后一部分，`var2` 的值被赋给 `var1`，`var1` 的值被赋给 `var2`。Python 会同时对元组的各个部分求值。这样，`red robin` 变成了 `robin red`。

3. 元组转换成列表（及列表转换成元组）

```
# 将元组转换成列表,列表转换成元组
my_list = [1, 2, 3]
my_tuple = ('x', 'y', 'z')
print("Output #100: {}".format(tuple(my_list)))
print("Output #101: {}".format(list(my_tuple)))
```

最后，可以将元组转换成列表，也可以将列表转换成元组。这个功能和可以将一个元素转换成字符串的 `str` 函数很相似。要将一个列表转换成元组，将列表名称放在 `tuple()` 函数中即可。同样，要将一个元组转换成列表，将元组名称放在 `list()` 函数中即可。

如果想获得元组的更多信息，可以参考 Python 标准库（https://docs.python.org/3/library/index.html）。

1.4.7 字典

Python 中的**字典**本质上是包含各种带有唯一标识符的成对信息的列表。和列表一样，字典也广泛应用于各种商业分析。在商业分析中，可以用字典表示客户（以客户编码为键值），也可以用字典表示产品（以序列号或产品编号为键值），还可以用字典表示资产、销售量等。

在 Python 中，这样的数据结构称为字典，在其他编程语言中则称为**关联数组**、**键-值存储**和**散列值**。在商业分析中，列表和字典都是非常重要的数据结构，但是它们之间还存在着重要的区别，要想有效地使用字典，必须清楚这些区别。

- 在列表中，你可以使用被称为**索引**或**索引值**的连续整数来引用某个列表值。在字典中，要引用一个字典值，则可以使用整数、字符串或其他 Python 对象，这些统称为**字典键**。在唯一键值比连续整数更能反映出变量值含义的情况下,这个特点使字典比列表更实用。
- 在列表中，列表值是隐式排序的，因为索引是连续整数。在字典中，字典值则没有排序，因为索引不仅仅只是数值。你可以为字典中的项目定义排序操作，但是字典确实没有内置排序。
- 在列表中，为一个不存在的位置（索引）赋值是非法的。在字典中，则可以在必要的时候创建新的位置（键）。
- 因为没有排序，所以当你进行搜索或添加新值时，字典的响应时间更快（当你插入一个新项目时，计算机不需要重新分配索引值）。当处理的数据越来越多时，这是一个重要的考虑因素。

因为字典在商业应用中使用广泛、灵活性高、作用突出，所以掌握如何在 Python 中使用字典是极其重要的。下面的示例代码演示了最常用的和最有效的用于处理字典的函数和操作符的使用方法。

1. 创建字典

```
# 使用花括号创建字典
# 用冒号分隔键-值对
# 用len()计算出字典中键-值对的数量
empty_dict = { }
a_dict = {'one':1, 'two':2, 'three':3}
print("Output #102: {}".format(a_dict))
print("Output #103: a_dict has {!s} elements".format(len(a_dict)))
another_dict = {'x':'printer', 'y':5, 'z':['star', 'circle', 9]}
print("Output #104: {}".format(another_dict))
print("Output #105: another_dict also has {!s} elements"\
.format(len(another_dict)))
```

这个示例展示了创建字典的方法。要创建一个空字典，将字典名称放在等号左侧，在等号右侧放上一对花括号即可。

示例中的第二个字典 `a_dict` 演示了向字典中添加键和值的一种方法。`a_dict` 说明键和值

是用冒号隔开的，花括号之间的键-值对则是用逗号隔开的。字典键是用单引号或双引号围住的字符串，字典值可以是字符串、数值、列表、其他字典或其他 Python 对象。在 a_dict 中，字典值是整数，但是在 another_dict 中，字典值是字符串、数值和列表。最后，这个示例说明了 len 函数返回的是字典中键-值对的个数。

2. 引用字典中的值

```
# 使用键来引用字典中特定的值
print("Output #106: {}".format(a_dict['two']))
print("Output #107: {}".format(another_dict['z']
```

要引用字典中一个特定的值，需要使用字典名称、一对方括号和一个特定的键值（一个字符串）。在这个示例中，a_dict['two'] 的结果是整数 2，another_dict['z'] 的结果是列表 ['star', 'circle', 9]。

3. 复制

```
# 使用copy()复制一个字典
a_new_dict = a_dict.copy()
print("Output #108: {}".format(a_new_dict))
```

要复制一个字典，先在字典名称后面加上 copy 函数，然后将这个表达式赋给一个新的字典即可。在这个示例中，a_new_dict 是字典 a_dict 的一个副本。

4. 键、值和项目

```
# 使用keys()、values()和items()
# 分别引用字典中的键、值和键-值对
print("Output #109: {}".format(a_dict.keys()))
a_dict_keys = a_dict.keys()
print("Output #110: {}".format(a_dict_keys))
print("Output #111: {}".format(a_dict.values()))
print("Output #112: {}".format(a_dict.items()))
```

要引用字典的键值，在字典名称后面加上 keys 函数即可。这个表达式的结果是包含字典键值的一个列表。要引用字典值，在字典名称后面加上 values 函数即可。这个表达式的结果是包含字典值的一个列表。

要想同时引用字典的键和值，在字典名称后面加上 items 函数即可。结果是一个列表，其中包含的是键-值对形式的元组。例如，a_dict.items() 的结果是 [('three', 3), ('two', 2), ('one', 1)]。在 1.4.8 节中会介绍如何使用 for 循环来对字典中的所有键和值进行解包和引用。

5. 使用in、not in和get

```
if 'y' in another_dict:
    print("Output #114: y is a key in another_dict: {}."\
.format(another_dict.keys()))
if 'c' not in another_dict:
    print("Output #115: c is not a key in another_dict: {}."\
.format(another_dict.keys()))
print("Output #116: {!s}".format(a_dict.get('three')))
print("Output #117: {!s}".format(a_dict.get('four')))
print("Output #118: {!s}".format(a_dict.get('four', 'Not in dict')))
```

这个示例展示了测试字典中是否存在某个键值的两种方法。第一种方法是使用 if 语句、in 或 not in 以及字典名称。使用 in、if 语句来测试 y 是否是 another_dict 中的一个键值。如果语句结果为真（也就是说如果 y 是 another_dict 中的一个键值），那么就执行 print 语句；否则，就不执行。这种 if in 和 if not in 语句经常用于测试是否存在某个键值，和其他语句一起使用时，还可以为字典添加新的键值。本书后续章节会有添加键值的示例。

> **缩进的作用**
>
> 你应该注意到 if 语句后面的行是缩进的。Python 使用缩进来表示一个语句是否属于一个逻辑上的"块"，也就是说，如果 if 语句判断为 True，那么 if 语句后面所有缩进的代码都要被执行，然后 Python 解释器再继续下一步工作。你会发现这种缩进还会在随后讨论的其他逻辑块中出现，现在你需要记住的是，在 Python 中缩进是有明确意义的，你必须遵循这个原则。如果你使用 IDE 或者像 Sublime Text 这样的编辑器，软件会帮助你设置每次使用制表符都相当于按了一定次数的空格键；如果你使用像 Notepad 一样的普通文本编辑器，那么就要注意使用同样数目的空格表示同一级别的缩进（一般使用 4 个空格）。
>
> 最后需要注意的是，有时候文本编辑器会使用制表符代替空格，这样程序看上去没有问题，但还是会收到一条错误信息（像"Unexpected indent in line 37"），这就会令程序员非常抓狂。尽管有这个问题，一般来说，Python 使用缩进还是会使代码更加清晰易读，因为你可以轻松地理解程序的逻辑过程。

第二种测试具体键值的方式是使用 get 函数，这个函数也可以按照键值取得相应的字典值。和前一种测试键值的方式不同，如果字典中存在这个键，get 函数就返回键值对应的字典值；如果字典中不存在这个键，则返回 None。此外，get 函数还可以使用第二个参数，表示如果字典中不存在键值时函数的返回值。通过这种方式，如果字典中不存在该键值，可以返回除 None 之外的一些其他内容。

6. 排序

```
# 使用sorted()对字典进行排序
# 要想对字典排序的同时不修改原字典
# 先复制字典
print("Output #119: {}".format(a_dict))
dict_copy = a_dict.copy()
ordered_dict1 = sorted(dict_copy.items(), key=lambda item: item[
print("Output #120 (order by keys): {}".format(ordered_dict1))
ordered_dict2 = sorted(dict_copy.items(), key=lambda item: item[1])
print("Output #121 (order by values): {}".format(ordered_dict2))
ordered_dict3 = sorted(dict_copy.items(), key=lambda x: x[1], reverse=True)
print("Output #122 (order by values, descending): {}".format(ordered_dict3))
ordered_dict4 = sorted(dict_copy.items(), key=lambda x: x[1], reverse=False)
print("Output #123 (order by values, ascending): {}".format(ordered_dict4))
```

这个示例展示了如何使用不同方式对字典进行排序。本节开头就说过，字典没有隐含排序，但是，你可以使用前面的代码片段对一个字典对象进行排序。可以按照字典的键或字典值来排序，如果这些值是数值型的，排序方式可以是升序，也可以是降序。

这个示例中使用了 copy 函数来为字典 a_dict 制作一个副本，副本的名称为 dict_copy。为字典制作副本确保了原字典 a_dict 不会被修改。下一行代码中包含了 sorted 函数、一个由 items 函数生成的元组列表和一个作为 sorted 函数关键字的 lambda 函数。

这行代码比较复杂，可以先将它分解一下。这行代码的目的是对 items 函数生成的键-值元组列表按照某种规则进行排序。这种规则就是 key，它相当于一个简单的 lambda 函数。（lambda 函数是一个简单函数，在运行时返回一个表达式。）在这个 lambda 函数中，item 是唯一的参数，表示由 items 函数返回的每个键-值元组。冒号后面是要返回的表达式，这个表达式是 item[0]，即返回元组中的第一个元素（也就是字典键值），用作 sorted 函数的关键字。简而言之，这行代码的意义是：将字典中的键-值对按照字典键值升序排序。下一个 sorted 函数使用 item[1] 而不是 item[0]，所以这行代码按照字典值对键-值对进行升序排序。

最后两行代码中的 sorted 函数与它们前面一行代码中的 sorted 函数很相似，因为这 3 个 sorted 函数都使用字典值作为排序关键字。又因为这个字典的值是数值型的，所以可以按照升序或降序对其进行排序。最后两种排序方式展示了如何使用 sorted 函数的 reverse 参数来设定排序结果是升序还是降序。reverse=True 对应降序，所以键-值对按照字典值以降序排序。

要想获得更多关于字典的信息，请参考 Python 标准库（https://docs.python.org/3/library/index.html）。

1.4.8 控制流

控制流元素非常重要，因为可以在程序中包含有意义的业务逻辑。很多商务处理和分析依赖于业务逻辑，例如“如果客户的花费超过一个具体值，那么就怎样怎样”或“如果销售额属于 A 类，则编码为 *X*，如果销售额属于 B 类，则编码为 *Y*，否则编码为 *Z*。”这些逻辑语句在代码中可以用控制流元素来表示。

Python 提供了若干种控制流元素，包括 if-elif-else 语句、for 循环、range 函数和 while 循环。正如它们的名字所示，if-else 语句提供的逻辑为“如果这样，那么就做那个，否则做些别的事情”。else 代码块并不是必需的，但可以使你的代码更加清楚。for 循环可以使你在一系列值之间进行迭代，这些值可以是列表、元组或字符串。你可以使用 range 函数与 len 函数一起作用于列表，生成一个索引序列，然后用在 for 循环中。最后，只要 while 条件为真，while 循环会一直执行内部的代码。

1. if-else

```
# if-else语句
x = 5
if x > 4 or x != 9:
    print("Output #124: {}".format(x))
else:
    print("Output #124: x is not greater than 4")
```

第一个示例展示了一个简单的 if-else 语句。if 条件判断 x 是否大于 4 或者 x 是否不等于 9（!= 操作法表示“不等于”）。通过使用 or 操作符，在找到一个表达式为 True 时就停止

判断。在这个例子中，x 等于 5，5 大于 4，所以 x != 9 是不用判断的。第一个条件 x > 4 为真，所以 print x 被执行，打印结果为数值 5。如果 if 代码块中没有一个条件为真，那么就执行 else 代码块中的 print 语句。

2. if-elif-else

```
# if-elif-else语句
if x > 6:
    print("Output #125: x is greater than six")
elif x > 4 and x == 5:
    print("Output #125: {}".format(x*x))
else:
    print("Output #125: x is not greater than 4")
```

第二个示例展示了一个稍微复杂一点的 if-elif-else 语句。同前一个示例一样，if 代码块测试 x 是否大于 6。如果这个条件为真，那么停止判断，执行相应的 print 语句。实际上，5 不大于 6，所以使用下面的 elif 语句继续判断。这个语句测试 x 是否大于 4 并且 x 是否等于 5。使用 and 操作符进行的判断在找到一个表达式为 False 时就停止。在这个例子中，x 等于 5，5 大于 4，并且求的 x 值是 5，所以执行 print x*x，打印结果为数值 25。因为这里已经使用了等号作为对象赋值操作符，所以用两个等号（==）判断是否相等。如果 if 和 elif 代码块都不为真，那么就执行 else 代码块中的 print 语句。

3. for循环

```
y = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', \
'Nov', 'Dec']
z = ['Annie', 'Betty', 'Claire', 'Daphne', 'Ellie', 'Franchesca', 'Greta', \
'Holly', 'Isabel', 'Jenny']

print("Output #126:")
for month in y:
     print("{!s}".format(month))

print("Output #127: (index value: name in list)")
for i in range(len(z)):
     print("{0!s}: {1:s}".format(i, z[i]))

print("Output #128: (access elements in y with z's index values)")
for j in range(len(z)):
     if y[j].startswith('J'):
         print("{!s}".format(y[j]))

print("Output #129:")
for key, value in another_dict.items():
     print("{0:s}, {1}".format(key, value))
```

这 4 个 for 循环示例演示了如何使用 for 循环在序列中迭代。在本书后面的章节，以及一般的商业应用中，这种功能都非常重要。第一个 for 循环示例展示了基本语法，即 for variable in sequence，做某事。variable 是一个临时占位符，表示序列中的各个值，并且只在 for 循环中有意义。在这个示例中，变量名为 month。sequence 是你要进行迭代的序列的名称。同样在这个示例中，序列名为 y，是一个月份列表。因此，这个示例的意义是："对于 y 中的每个值，打印出这个值。"

第二个 for 循环示例展示了如何使用 range 函数和 len 函数的组合生成一个可以在 for 循环中使用的索引值序列。为了弄清楚复合函数之间的相互作用，可以仔细地分析一下。len 函数返回列表 z 中元素的个数，这里的个数是 10。然后 range 函数生成了一系列整数，是从 0 开始直到比 len 函数的结果少 1 的整数，在这个例子中，就是 0~9 的整数。因此，for 循环的意义就是："对于 0~9 的整数序列中的一个整数 i，打印出整数 i，再打印一个空格，然后打印出列表 z 中索引值为 i 的元素值。"在本书的很多示例中，你都会看到 range 函数和 len 函数的组合用在 for 循环中，因为这种组合在很多商业应用中是非常有用的。

第三个 for 循环示例展示了如何使用从一个序列中生成的索引值来引用另一个序列中具有同样索引值的元素，也说明了如何在 for 循环中包含 if 语句来说明业务逻辑。这个示例又一次使用 range 函数和 len 函数生成了列表 z 的索引值。然后，if 语句测试列表 y 中具有这些索引值的元素（y[0]='Jan', y[1]='Feb', …, y[9]='Oct'）是否以大写字母 J 开头。

最后一个 for 循环展示了在字典的键和值之间进行迭代和引用的方法。在 for 循环的第一行中，items 函数返回字典的键–值元组。for 循环中的 key 和 value 变量依次捕获这些值。for 循环中的 print 语句在每一行打印出一个键–值对，键和值之间以逗号隔开。

4. 简化for循环：列表、集合与字典生成式

列表、集合与字典**生成式**是 Python 中一种简化的 for 循环写法。列表生成式出现在方括号内，集合生成式与字典生成式则出现在花括号内。所有的生成式都包括条件逻辑（例如：if-else 语句）。

列表生成式。下面的示例展示了如何使用列表生成式来从一个列表集合中筛选出符合特定条件的列表子集：

```
# 使用列表生成式选择特定的行
my_data = [[1,2,3], [4,5,6], [7,8,9]]
rows_to_keep = [row for row in my_data if row[2] > 5]
print("Output #130 (list comprehension): {}".format(rows_to_keep))
```

在这个示例中，列表生成式的意义是：对于 my_data 中的每一行，如果这行中索引位置 2 的值（即第三个值）大于 5，则保留这一行。因为 6 和 9 都大于 5，所以 rows_to_keep 中的列表子集为 [4, 5, 6] 和 [7, 8, 9]。

集合生成式。下面的示例展示了如何使用集合生成式来从一个元组列表中选择出特定的元组集合：

```
# 使用集合生成式在列表中选择出一组唯一的元组
my_data = [(1,2,3), (4,5,6), (7,8,9), (7,8,9)]
set_of_tuples1 = {x for x in my_data}
print("Output #131 (set comprehension): {}".format(set_of_tuples1))
set_of_tuples2 = set(my_data)
print("Output #132 (set function): {}".format(set_of_tuples2))
```

在这个示例中，集合生成式的意义是：对于 my_data 中的每个元组，如果它是一个唯一的元组，则保留这个元组。你可以称这个表达式为集合生成式而不是列表生成式，因为表达式中是花括号，不是方括号，而且它也不是字典生成式，因为它没有使用键–值对这样的语法。

这个示例中的第二个 print 语句说明你可以通过 Python 内置的 set 函数达到和集合生成式同样的效果。在这个例子中，使用内置的 set 函数更好，因为它比集合生成式更精炼，而且更易读。

字典生成式。下面的示例展示了如何使用字典生成式来从一个字典中筛选出满足特定条件的键-值对子集：

```
# 使用字典生成式选择特定的键-值对
my_dictionary = {'customer1': 7, 'customer2': 9, 'customer3': 11}
my_results = {key : value for key, value in my_dictionary.items() if \
value > 10}
print("Output #133 (dictionary comprehension): {}".format(my_results))
```

在这个例子中，字典生成式的意义为：对于 my_dictionay 中的每个键-值对，如果值大于 10，则保留这个键-值对。因为值 11 大于 10，所以保留在 my_results 中的键-值对为 {'customer3':11}。

5. while循环

```
print("Output #134:")
x = 0
while x < 11:
    print("{!s}".format(x))
    x += 1
```

这个示例展示了如何使用 while 循环来打印 0~10 的整数。x = 0 将变量 x 初始化为 0。然后 while 循环判断 x 是否小于 11。因为 x 小于 11，所以 while 循环在一行中打印出 x 值，然后将 x 的值增加 1。接着 while 循环继续判断 x（此时为 1）是否小于 11。因为确实小于 11，所以继续执行 while 循环内部的语句。这个过程一直以这种方式继续，直到 x 从 10 增加到 11。这时，while 循环继续判断 x 是否小于 11，表达式结果为假，while 循环内部语句不再被执行。

while 循环适合于知道内部语句会被执行多少次的情况。更多时候，你不太确定内部语句需要执行多少次，这时就应该使用 for 循环。

6. 函数

在一些情况下，你会发现自己编写函数比使用 Python 内置函数和安装别人开发的模块更方便有效。举例来说，如果你发现总是在不断重复地书写同样的代码片段，那么就应该考虑将这个代码片段转换为函数。某些情况下，函数可能已经存在于 Python 基础模块或“可导入”的模块中了。如果函数已经存在，就应该使用这些开发好并已经通过了大量测试的函数。但是，有些情况下，你需要的函数不存在或不可用，这时就需要你自己创建函数。

要在 Python 中创建函数，需要使用 def 关键字，并在后面加上函数名称和一对圆括号，然后再加上一个冒号。组成函数主体的代码需要缩进。最后，如果函数需要返回一个或多个值，可以使用 return 关键字来返回函数结果供程序使用。下面的示例展示了在 Python 中创建和使用函数的方法：

```
# 计算一系列数值的均值
def getMean(numericValues):
    return sum(numericValues)/len(numericValues) if len(numericValues) > 0
```

```
        else float('nan')

my_list = [2, 2, 4, 4, 6, 6, 8, 8]
print("Output #135 (mean): {!s}".format(getMean(my_list))
```

这个示例展示了如何创建函数来计算一系列数值的均值。函数名为 getMean。圆括号之间的短语表示要传入函数中的数值序列，这是一个仅在**函数作用范围内**有意义的变量。在函数内部，由序列的总和除以序列中数值的个数计算出序列的均值。此外，还可以使用 if-else 语句来检验序列中是否包含数值。如果确实包含数值，函数返回序列均值。如果不包含数值，函数返回 nan（即：非数值）。如果省略了 if-else 语句，而且序列中正好没有任何数值的话，程序就会抛出一个除数为 0 的错误。最后，使用 return 关键字返回函数结果供程序使用。

在这个示例中，my_list 包含了 8 个数值。my_list 被传入 getMean() 函数中。8 个数值的总和为 40，40 除以 8 等于 5。所以，print 语句打印出整数 5。

就像你猜测的那样，mean 函数已经存在了。例如，NumPy 中就有一个 mean 函数。所以，你可以通过导入 NumPy，使用它里面的 mean 函数得到同样的结果：

```
import numpy as np
print np.mean(my_list)
```

再说一次，如果在 Python 基础程序或可导入模块中，已经存在你需要的函数，就应该使用这些开发好的并已经通过了大量测试的函数。使用 Google 或 Bing 来搜索“< 你需要的功能描述 > Python function”可以帮助你找到想要的 Python 函数。但是，如果你想完成业务过程中特有的任务，知道如何去创建函数还是非常重要的。

7. 异常

编写一个强壮稳健的程序的一个重要方面就是有效地处理错误和异常。在编写程序时，你可能会隐含地假设程序要处理的数据类型和数据结构，如果有数据违反了你的假设，就会使程序抛出错误。

Python 中包含了若干种内置的异常对象。常用的异常包括 IOError、IndexError、KeyError、NameError、SyntaxError、TypeError、UnicodeError 和 ValueError。你可以在网上获得更多的异常信息，参见 Python 标准库中的“Built-in Exceptions”那一节（http://docs.python.org/3/library/exceptions.html）。你可以使用 try-except 来构筑处理错误信息的第一道防线，即使数据不匹配，你的程序还可以继续运行。

下面展示了两种使用 try-except 代码块来有效地捕获和处理异常的方法（一种比较短，另一种比较长）。这两个示例修改了上一节的函数示例，来说明如何使用 try-except 代码块代替 if 语句处理空列表的情况。

8. try-except

```
# 计算一系列数值的均值
def getMean(numericValues):
    return sum(numericValues)/len(numericValues)
my_list2 = [ ]

# 简单形式
```

```
try:
    print("Output #138: {}".format(getMean(my_list2)))
except ZeroDivisionError as detail:
    print("Output #138 (Error): {}".format(float('nan')))
    print("Output #138 (Error): {}".format(detail))
```

在这种处理异常的方法中，函数 getMean() 中没有检验序列是否包含数值的 if 语句。如果序列是空的，就像列表 my_list2 一样，那么调用这个函数会导致一个异常 ZeroDivisionError。

要想使用 try-except 代码块，需要将你要执行的代码放在 try 代码块中，然后，使用 except 代码块来处理潜在的错误并打印出错误信息来帮助你理解程序错误。在某些情况下，异常具有特定的值。你可以通过在 except 行中加上 as 短语来引用异常值，然后打印出你为异常指定的变量。因为 my_list2 中不包含任何数值，所以执行 except 代码块，打印出 nan 和 Error: float division by zero。

9. try-except-else-finally

```
# 完整形式
try:
    result = getMean(my_list2)
except ZeroDivisionError as detail:
    print "Output #142 (Error): " + str(float('nan'))
    print "Output #142 (Error):", detail
else:
    print "Output #142 (The mean is):", result
finally:
    print "Output #142 (Finally): The finally block is executed every time"
```

这个完整形式的异常处理方法除了 try 和 except 代码块，还包含 else 和 finally 代码块。如果 try 代码块成功执行，则会接着执行 else 代码块。因此，如果传递给 try 代码块中的 getMean() 函数的数值序列中包含任意数值，那么这些数值的均值就会被赋给 try 代码块中的变量 result，然后接着执行 else 代码块。例如，如果这里使用程序代码 +my_list1+，就会打印出 The mean is: 5.0。因为 my_list2 不包含任何数值，所以执行 except 代码块，打印出 nan 和 Error: float division by zero。然后，总是执行 finally 模块，打印出 The finally block is executed every time。

1.5 读取文本文件

数据几乎无一例外地是被保存在文件中的。这些文件可能是文本文件、CSV 文件、Excel 文件或其他类型的文件。知道如何访问此类文件以及从中读取数据是在 Python 中进行数据处理、加工与分析的前提。当完成了一个每秒钟可以处理很多文件的程序时，与手动一个个地处理文件相比，你会真正体会到写程序的好处。

你需要告诉 Python，脚本要处理何种类型的文件。你可以在程序中写死文件名称，但是如果这样的话，就不能使用这个程序处理多个不同的文件了。能读取多个不同文件的方法是，在命令行窗口或终端窗口的命令行中，在 Python 脚本的名字后面加上完整的文件路径名。要使用这种方法，需要在脚本开始时导入内置的 sys 模块。在脚本上方加上 import

sys 语句之后，就可以在脚本中使用 sys 模块提供的所有功能了：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
import re
from datetime import date, time, datetime, timedelta
from operator import itemgetter
import sys
```

导入了 sys 模块之后，你就可以使用 argv 这个列表变量了。这个变量捕获了传递给 Python 脚本的**命令行参数**列表，即你在命令行中的所有输入，包括你的脚本名称。和任何其他列表一样，argv 也有索引。argv[0] 就是脚本名称，argv[1] 是命令行中传递给脚本的第一个附加参数，在这个例子中，就是 first_script.py 将要读取的文件路径名。

1.5.1 创建文本文件

要读取一个文本文件，首先要创建它。要创建文本文件，需执行以下步骤。

(1) 打开 Spyder IDE 或一个文本编辑器（例如：Windows 系统下的 Notepad、Notepad++、Sublime Text；macOS 系统下的 TextMate、TextWrangler、Sublime Text）。

(2) 在文本文件中写入下面 6 行（参见图 1-10）：

```
I'm
already
much
better
at
Python.
```

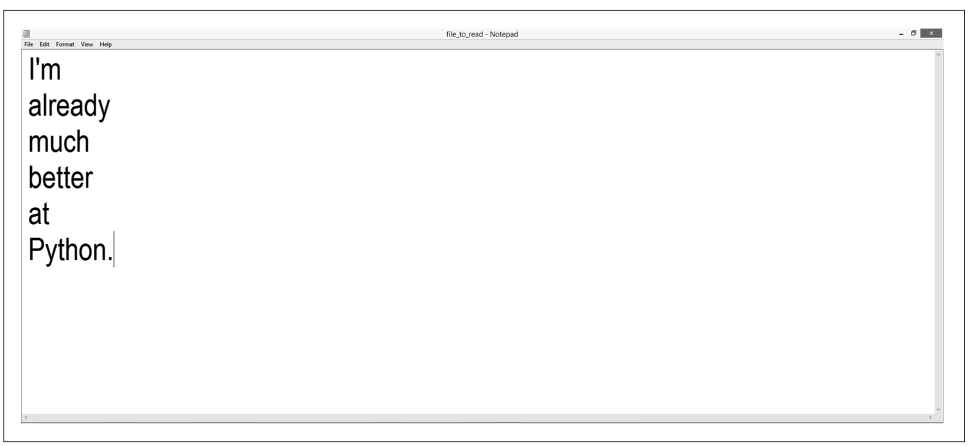


图 1-10：Notepad++ 中的文本文件 file_to_read.txt（Windows）

(3) 将文件保存在桌面上，文件名为 file_to_read.txt。

(4) 将下面几行代码添加到 first_script.py 的下方：

```
# 读取文件
```

```
# 读取单个文本文件
input_file = sys.argv[1]

print "Output #143: "
filereader = open(input_file, 'r')
for row in filereader:
    print row.strip()
filereader.close()
```

示例中的第一行代码使用 `sys.argv` 列表捕获了要读取的文件的路径名，并将路径名赋给变量 `input_file`。第二行代码创建了一个文件对象 `filereader`，其中包含了以 `r` 模式（只读模式）打开的 `input_file` 文件中的各个行。下一行中的 `for` 循环每次读取 `filereader` 对象中的一行。`for` 循环内部的 `print` 语句打印出每一行，并且在打印之前用 `strip` 函数去掉每一行两端的空格、制表符和换行符。最后一行代码在输入文件中的所有行都被读取并打印到屏幕后，关闭 `filereader` 对象。

(5) 重新保存 first _script.py。

(6) 要读取刚才创建的文本文件，输入下面的命令，如图 1-11 所示，然后按回车键：

```
python first_script.py file_to_read.txt
```

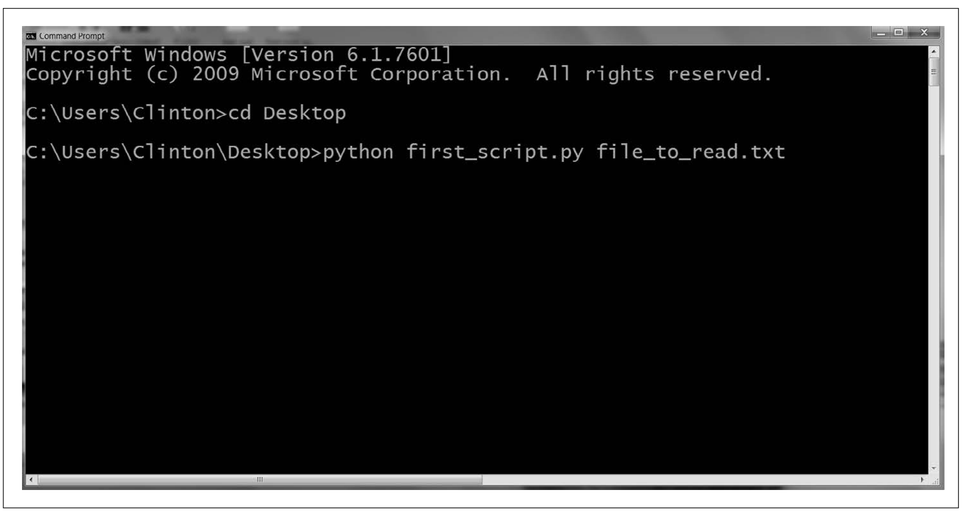


图 1-11：Python 脚本和它要在命令行窗口中处理的文本文件

这样，你就在 Python 中读取了一个文本文件。你会看到下面的内容被打印到屏幕上，在以前的输出之后（图 1-12）：

```
I'm
already
much
better
at
Python.
```



图 1-12：first_script.py 的输出，在命令行窗口中处理文本文件

1.5.2　脚本和输入文件在同一位置

因为 first_script.py 和 file_to_read.txt 在同一位置，即都在桌面上，所以简单地输入 `python first_script.py file_to_read.txt` 是可以的。如果文本文件和脚本不在同一位置，就需要输入文本文件的完整路径名，这样脚本才能知道去哪里寻找这个文件。

例如，如果文本文件在你的 Documents 文件夹中，而不是在桌面上，那么你可以在命令行中使用下面的路径名来从其所处位置读取文本文件：

```
python first_script.py "C:\Users\[Your Name]\Documents\file_to_read.txt"
```

1.5.3　读取文件的新型语法

前面讲的用来创建 `filereader` 对象的那行代码是创建文件对象的传统方法。这种方法没有什么问题，但是它使文件对象一直处于打开状态，直到使用 `close` 函数明确地关闭或直到脚本结束。尽管这种做法一般没有问题，但不够清晰，还被证明在更复杂的脚本中会导致错误。从 Python 2.5 开始，你可以使用 `with` 语句来创建文件对象。这种语法在 `with` 语句结束时会自动关闭文件：

```
input_file = sys.argv[1]
print("Output #144:")
with open(input_file, 'r', newline='') as filereader:
for row in filereader:
    print("{}".format(row.strip()))
```

你可以看到，使用 `with` 语句的版本与前一个版本非常相似，但是它不需调用 `close` 函数来关闭 `filereader` 对象。

这个示例演示了如何使用 `sys.argv` 来访问并打印一个文本文件中的内容。这是一个简单的示例，但在后面的示例中，要以此为基础访问其他类型的文件，或一次访问多个文件，并

向输出文件中写入内容。

下一节介绍 glob 模块，它让你能够通过几行代码读取和处理多个输入文件。glob 模块之所以功能强大，是因为它处理的是文件夹（也就是说，它处理目录，不是单个的文件），所以将前面读取文件的代码删除或注释掉，这样就可以使 argv[1] 指向一个文件夹，而不是一个文件了。将代码注释掉就是在你希望计算机忽略掉的代码前面加上一个井号，所以当你结束注释时，first_script.py 文件就应该像下面这样：

```
## 读取一个文本文件(旧方法）##
#input_file = sys.argv[1]
#print("Output #143:")
#filereader = open(input_file, 'r', newline='')
#for row in filereader:
# print("{}".format(row.strip()))
#filereader.close()
## 读取一个文本文件(新方法）##
#input_file = sys.argv[1]
#print("Output #144:")
#with open(input_file, 'r', newline='') as filereader:
# for row in filereader:
# print("{}".format(row.strip()))
```

做完这些修改之后，你就可以添加下一节要讨论的 glob 代码来处理多个文件了。

1.6 使用glob读取多个文本文件

在很多商业应用中，需要对多个文件进行同样的或相似的处理。例如，你可能会从多个文件中选择数据子集，根据多个文件计算像总计和均值这样的统计量，或根据来自于多个文件的数据子集计算统计量。当文件数量增加时，手动处理文件的可能性会减小，出错的概率会增加。

读取多个文件的一种方法是在命令行中将包含输入文件目录的路径名写在 Python 脚本名称之后。要使用这种方法，你需要在脚本开头导入内置的 os 模块和 glob 模块。在脚本上方添加了 import os 和 import glob 语句之后，你就可以使用 os 模块和 glob 模块提供的所有功能了：

```
#!/usr/bin/env python3
from math import exp, log, sqrt
import re
from datetime import date, time, datetime, timedelta
from operator import itemgetter
import sys
import glob
import os
```

当导入了 os 模块之后，你就可以使用它提供的若干种路径名函数了。例如，os.path.join 函数可以巧妙地将一个或多个路径成分连接在一起。glob 模块可以找出与特定模式相匹配的所有路径名。os 模块和 glob 模块组合在一起使用，可以找出符合特定模式的某个文件夹下面的所有文件。

要读取多个文件，需要再创建一个文本文件。

创建另一个文本文件

(1) 打开 Spyder IDE 或一个文本编辑器（例如：Windows 系统下的 Notepad、Notepad++、Sublime Text；macOS 系统下的 TextMate、TextWrangler、Sublime Text）。

(2) 在文本文件中写入下面 8 行（参见图 1-13）：

```
This
text
comes
from
a
different
text
file.
```



图 1-13：Notepad++ 中的文本文件 another_file_to_read.txt

(3) 将文件保存在桌面上，文件名为 another_file_to_read.txt。

(4) 将下面几行代码添加到 first_script.py 的下方：

```
# 读取多个文本文件
print("Output #145:")
inputPath = sys.argv[1]
for input_file in glob.glob(os.path.join(inputPath,'*.txt')):
    with open(input_file, 'r', newline='') as filereader:
        for row in filereader:
            print("{}".format(row.strip()))
```

这个示例中的第一行代码与读取单个文本文件示例中的代码非常相似，只是在这个示例中，要提供一个目录路径名，而不是一个文件路径名。这里，要提供的路径指向包含了两个文本文件的目录。

第二行代码是 `for` 循环，使用 `os.path.join` 函数和 `glob.glob` 函数来找出符合特定模式的某个文件夹下面的所有文件。指向这个文件夹的路径包含在变量 `inputpath` 中，这个变量将在命令行中被提供。`os.path.join` 函数将这个文件夹路径和这个文件夹中所有符合特定模式的文件名连接起来，这种特定模式可以由 `glob.glob` 函数扩展。这个示例使用的是模式 `*.txt` 来匹配由 .txt 结尾的所有文件名。因为这是一个 `for` 循环，所以这行中其余的代码你应该很熟悉了。`input_file` 是一个占位符，表示由 `glob.glob` 函数生成的列表中的每个文件。这行代码的意义就是：对于匹配文件列表中的每个文件，做下面的操作……

余下的代码和读取单个文件的代码非常相似。以只读方式打开 `input_file` 变量，然后创建一个 `filereader` 对象。对于 `filereader` 对象中的每一行，除去行两端的空格、制表符和换行符，然后打印这一行。

(5) 重新保存 first _script.py。

(6) 要读取这些文本文件，输入以下代码，如图 1-14 所示，然后按回车键：

```
python first_script.py "C:\Users\[Your Name]\Desktop"
```

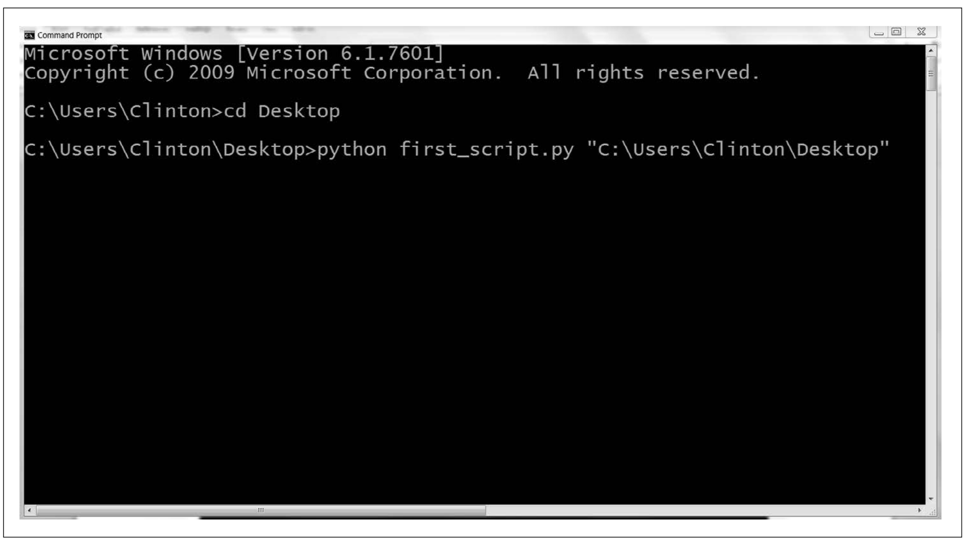


图 1-14：命令行窗口中的 Python 脚本和指向包含文本文件的桌面文件夹的路径

这样，你就在 Python 中读取了多个文本文件。你会看到以下内容被打印到屏幕上，在以前的输出之后（图 1-15）：

```
This
text
comes
from
a
different
text
file.
```

```
I'm
already
much
better
at
Python.
```

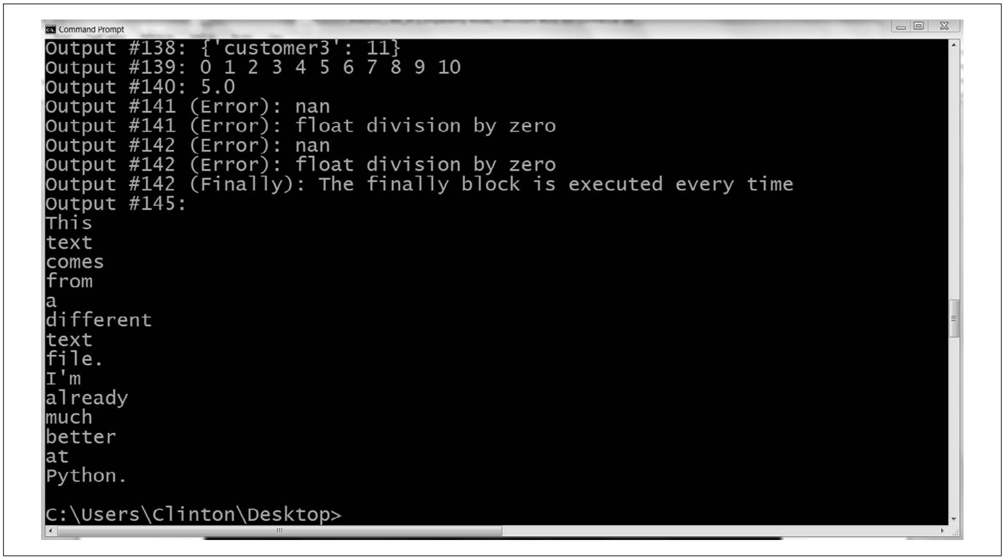


图 1-15：first_script.py 的输出，在命令行窗口中处理多个文本文件

学会这项技术的一个巨大好处是它可以规模化扩展。这个示例只是处理两个文本文件，但是它可以轻松地扩展为处理几十、几百或者几千甚至更多的文件。学习了如何使用 `glob.glob` 函数，仅花费手动处理的一小部分时间，就可以处理非常非常多的文件。

1.7　写入文本文件

迄今为止，大多数示例还是使用 `print` 语句将输出发送到命令行窗口或终端窗口。当你在调试程序，或者在检查输出的准确度时，将输出打印到屏幕上是有意义的。但是，在很多情况下，只要你能确定输出是正确的，就会需要将输出写入文件，以进行更进一步的分析、报告和存储。

Python 提供了两种简单的方法来将输出写入文本文件和分隔符文件。`write` 方法可将单个字符串写入一个文件，`writelines` 方法可将一系列字符串写入一个文件。下面的两个示例使用 `range` 函数和 `len` 函数跟踪一个列表中的索引值，以将分隔符放在各个列表值之间，并在最后一个列表值后面放上一个换行符。

1.7.1　向first_script.py添加代码

(1) 将下面各行代码添加到 first_script.py 的底部：

```
# 写入文件
# 写入一个文本文件
my_letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
max_index = len(my_letters)
output_file = sys.argv[1]
filewriter = open(output_file, 'w')
for index_value in range(len(my_letters)):
    if index_value < (max_index-1):
        filewriter.write(my_letters[index_value]+'\t')
    else:
        filewriter.write(my_letters[index_value]+'\n')
filewriter.close()
print "Output #146: Output written to file"
```

在这个例子中，变量 my_letters 是一个字符串列表。这里想把这些字母打印到一个文本文件中，每个字母之间用制表符分隔。这个示例中的难点是确保在字母之间以制表符分隔，并在最后一个字母后面放上一个换行符（不是制表符）。

为了知道什么时候到达最后一个字母，你需要跟踪列表中字母的索引值。len 函数用来计算出列表中字母的数量，所以 max_index 等于 10。在命令行窗口或终端窗口中，再次使用 sys.argv[1] 来在命令行中提供输出文件的路径名。创建一个文件对象 filewriter，但是打开方式不是只读，而是通过 w（可写）的方式打开。使用 for 循环在列表 my_letters 的各个值之间进行迭代，并使用 range 函数和 len 函数跟踪列表中各个字母的索引值。

if-else 语句可以使你对列表中的最后一个字母做出与前面那些字母不同的处理。if-else 语句是这样工作的：my_letters 包含 10 个元素，但是索引从 0 开始，所以各个字母的索引值分别是 0、1、2、3、4、5、6、7、8、9。因此，my_letters[0] 是 a，my_letters[9] 是 j。if 代码块判断索引值 x 是否小于 9，max_index – 1 或者是 10 – 1 = 9。直到列表中的最后一个字母，这个条件才为 True。因此，if 代码块的意义是：一直到列表中的最后一个字母，都向输出文件中写入字母，并在字母后面加一个制表符。当你到达了列表中的最后一个字母时，这个字母的索引值为 9，不大于 9，所以 if 代码块判断为 False，就执行 else 代码块。else 代码块中的 write 语句的意义是：向输出文件中写入最后一个字母，并在后面加一个换行符。

(2) 将前面读取多个文件的代码注释掉。

为了看到这些代码是如何工作的，这里需要写入一个文件然后查看输出。因为你又一次使用了 argv[1] 来确定输出文件的路径名，所以需要将前面的 glob 代码删除或注释掉，这样就可以使用 argv[1] 来确定输出文件了。如果选择注释掉前面的 glob 代码，那么 first_script.py 应该如下所示：

```
## 读取多个文本文件
#print("Output #145:")
#inputPath = sys.argv[1]
#for input_file in glob.glob(os.path.join(inputPath,'*.txt')):
#    with open(input_file, 'r', newline='') as #filereader:
#        for row in filereader:
#            print("{}".format(row.strip()))
```

(3) 重新保存 first_script.py。

(4) 要写入一个文本文件，输入下面的代码，如图 1-16 所示，然后按回车键：

```
python first_script.py "C:\Users\[Your Name]\Desktop\write_to_file.txt"
```

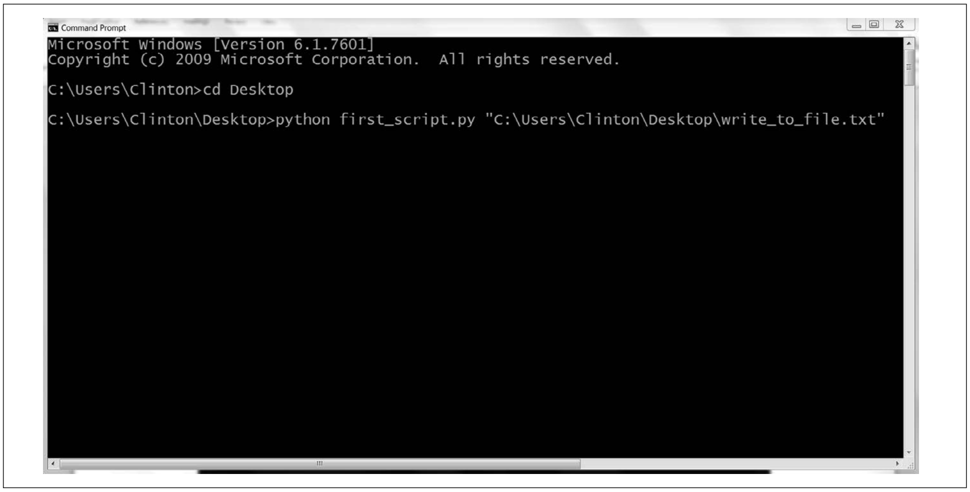


图 1-16：应该在命令行窗口中输入的 Python 脚本、文件路径和输出文件名

(5) 打开输出文件 write_to_file.txt。

这样，你就使用 Python 将输出写入了一个文本文件。在完成这些步骤之后，你不会在屏幕上看到新的输出；但是，如果你将所有打开的窗口最小化，就会看到桌面上有一个新的文本文件，名为 write_to_file.txt。这个文件中应该包含了列表 `my_letters` 中的字母，以制表符隔开，最后有一个换行符，如图 1-17 所示。

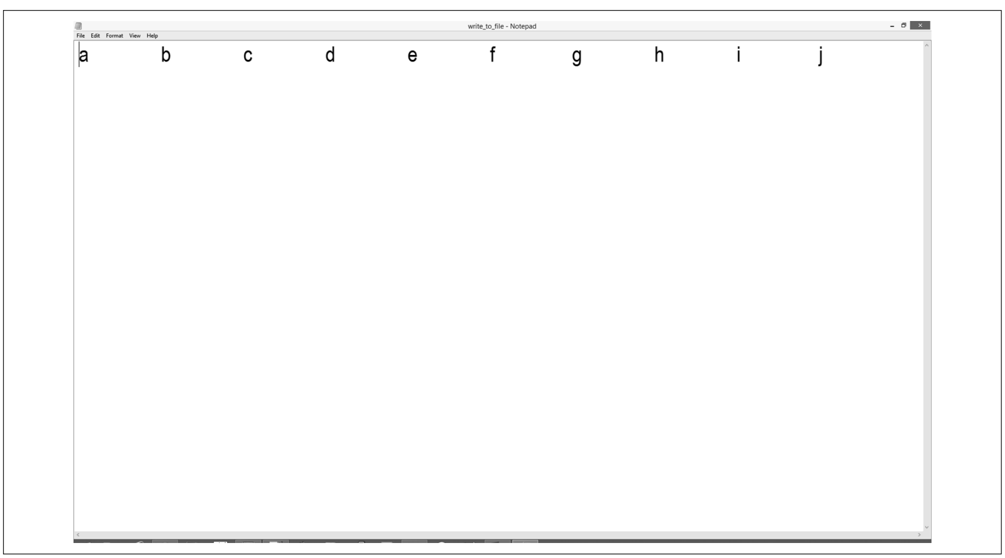


图 1-17：输出文件 write_to_file.txt，由 first_script.py 在桌面上创建

下一个示例与这个很相似，只是它演示了如何使用 `str` 函数来将元素转换为字符串，以便使用 `write` 函数将其写入一个文件。它还演示了使用 `'a'`（追加）方式将输出追加到一个已经存在的输出文件末尾的方法。

1.7.2 写入CSV文件

(1) 将下列各行代码添加到 first_script.py 的底部：

```
# 写入CSV文件
my_numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
max_index = len(my_numbers)
output_file = sys.argv[1]
filewriter = open(output_file, 'a')
for index_value in range(len(my_numbers)):
    if index_value < (max_index-1):
        filewriter.write(str(my_numbers[index_value])+',')
    else:
        filewriter.write(str(my_numbers[index_value])+'\n')
filewriter.close()
print "Output #147: Output appended to file"
```

这个示例与前面的示例非常相似，但是它说明了如何向已经存在的输出文件中追加内容，以及如何将列表中的非字符串数据转换成字符串，以便可以使用 `write` 函数来写入文件。在这个示例中，列表中的元素是整数。`write` 函数处理的是字符串，所以在你使用 `write` 函数将其写入输出文件之前，需要使用 `str` 函数将非字符串数据转换成字符串。

在使用 `for` 循环进行第一次迭代时，`str` 函数会向输出文件中写入一个 0，然后写入一个逗号。以这种方式继续写入列表中的其他数值，直到列表中的最后一个数值，这时执行 `else` 代码块，将最后一个数值写入输出文件，并在后面加上一个换行符，而不是逗号。

请注意在打开文件对象 `filewriter` 时，使用的是追加模式（`'a'`），而不是可写模式（`'w'`）。如果在命令行中提供了同样的输出文件名，那么这段代码的输出会被追加到 write_to_file.txt 文件中，在以前写入文件的内容之后。相反，如果使用可写方式打开 `filewriter` 对象，那么以前的输出会被删除，write_to_file.txt 文件中只会出现这段代码的输出。你会在本书后面的章节中看到使用追加方式打开文件的作用，这时你要处理多个文件，并将其中所有的数据追加到一个连接文件中。

(2) 重新保存 first_script.py。

(3) 要向文本文件中追加数据，输入以下命令然后按回车键：

```
python first_script.py "C:\Users\[Your Name]\Desktop\write_to_file.txt"
```

(4) 打开输出文件 write_to_file.txt。

这样，你就使用 Python 向文本文件中写入和追加了数据。在完成这些步骤之后，你不会在屏幕上看到新的输出；但是，如果你打开了 write_to_file.txt 文件，会看到文件中出现了新的一行，行中包括了 `my_numbers` 中的数值，以逗号隔开，并在末尾有一个换行符，如图 1-18 所示。

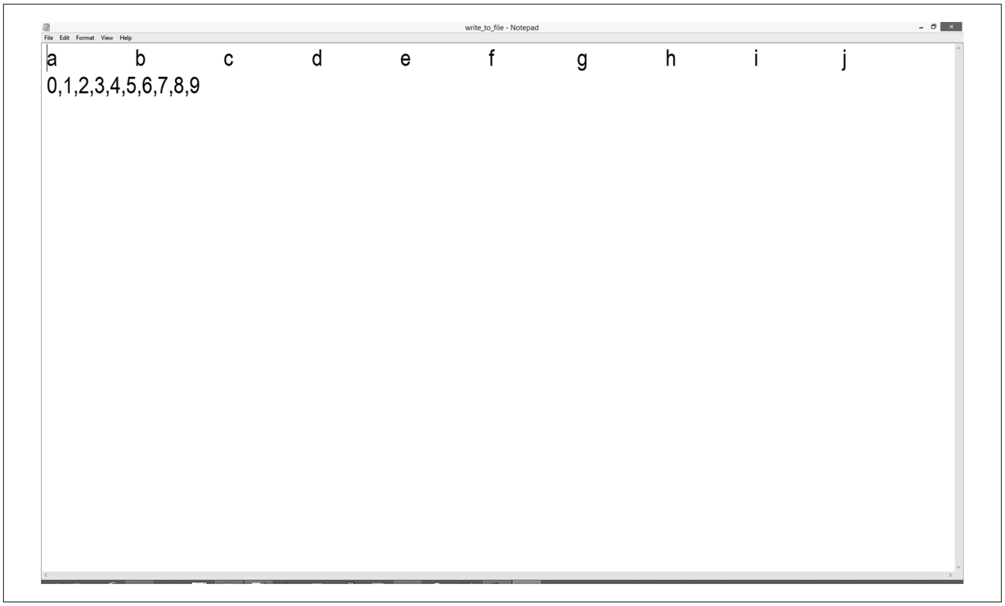


图 1-18：桌面上的输出文件 write_to_file.txt，由 first_script.py 追加了信息

最后，这个示例演示了一个写入 CSV 文件的有效方法。实际上，在前面的例子中，你将由制表符分隔的数据写入了输出文件，如果将制表符改为逗号，并且将输出文件命名为 write_to_file.csv 而不是 write_to_file.txt 的话，就可以创建一个 CSV 文件。

1.8 print语句

print 语句非常有助于程序调试。你已经看到，本章中很多示例程序都使用 print 语句作为输出。但是，你还可以在代码中临时添加 print 语句来检查中间结果。如果你的代码运行不了或者不能产生你需要的结果，那么可以在程序开头适当的地方添加 print 语句，看看最初的计算是否符合你的预期。如果符合，就继续检查下面的代码，看看它们是否按照你的预期工作。

从脚本开头进行检查，可以保证你找出第一个出错的地方并在检查余下的代码之前进行修复。本节的中心思想就是：不要吝啬在程序中使用 print 语句，它可以帮助你调试代码并保证代码正确。当你确信代码正确之后，完全可以将 print 语句注释掉或删除。

这一章介绍了很多基础知识，包括如何导入模块、基本的数据类型和与之相关的函数和方法、模式匹配、print 语句、日期处理、控制流、函数、异常、读取单个或多个文件，以及写入文本文件和分隔符文件。如果你一直跟随本章中的示例进行练习，那么你已经写了 500 多行 Python 代码了！

练习本章中示例代码的最大好处是，它们是编写更复杂的文件处理和数据操作程序的基础。熟练掌握了本章中的示例代码，你就可以理解并掌握本书后续章节介绍的各种技术。

1.9 本章练习

练习答案在附录 B 中。

(1) 创建一个新的 Python 脚本，在它里面创建 3 个不同的列表，将这 3 个列表相加，并使用 `for` 循环和定位索引（也就是 `range(len())`）在列表中循环，在屏幕上打印出列表的索引值和元素值。

(2) 创建一个新的 Python 脚本，在它里面创建两个同样长度的不同列表，其中一个列表包含具有唯一性的字符串。再创建一个空字典。使用 `for` 循环、定位索引和 `if` 语句检查字符串列表中的每个元素是否是字典中的键。如果不是，将这个元素作为字典键，将另一个列表中具有同样索引位置的元素作为字典值，把它们添加到字典中。最后在屏幕上打印出字典的键和值。

(3) 创建一个新的 Python 脚本，在它里面创建一个列表，其中的元素是具有相同长度的列表。修改 1.7.2 节中的代码，在列表的列表中循环，将每个列表中的值以逗号隔开的字符串形式打印在屏幕上，并在每个列表的最后一个值后面加上一个换行符。

第 2 章

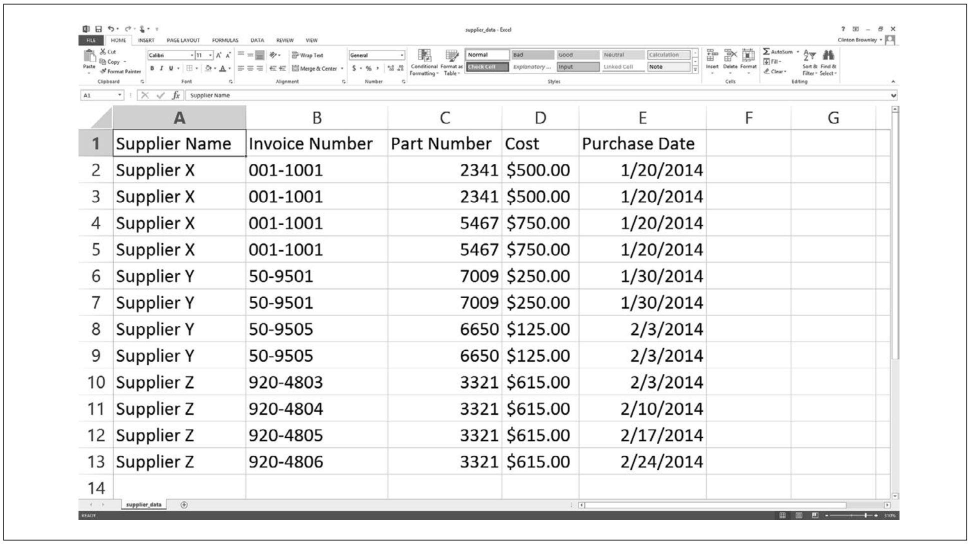
CSV文件

CSV（comma-separated value，逗号分隔值）文件格式是一种非常简单的数据存储与分享方式。CSV 文件将数据表格存储为纯文本，表格（或电子表格）中的每个单元格都是一个数值或字符串。与 Excel 文件相比，CSV 文件的一个主要优点是有很多程序可以存储、转换和处理纯文本文件；相比之下，能够处理 Excel 文件的程序却不多。所有电子表格程序、文字处理程序或简单的文本编辑器都可以处理纯文本文件，但不是所有的程序都能处理 Excel 文件。尽管 Excel 是一个功能非常强大的工具，但是当你使用 Excel 文件时，还是会被局限在 Excel 提供的功能范围内。CSV 文件则为你提供了非常大的自由，使你在完成任务的时候可以选择合适的工具来处理数据——如果没有现成的工具，那就使用 Python 自己开发一个！

当你使用 CSV 文件时，确实会失去某些 Excel 功能：在 Excel 电子表格中，每个单元格都有一个定义好的“类型”（数值、文本、货币、日期等），CSV 文件中的单元格则只是原始数据。幸好，Python 在识别不同数据类型方面相当聪明，第 1 章中已经展示了这一点。使用 CSV 文件的另一个问题是它只能保存数据，不能保存公式。但是，通过将数据存储（CSV 文件）和数据处理（Python 脚本）分离，你可以很容易地在不同数据集上进行加工处理。当数据存储和数据处理过程分开进行时，错误（不管是数据处理中的错误，还是数据存储中的错误）不但更容易被发现，而且更难扩散。

要使用 CSV 文件开始工作，需要先创建一个 CSV 文件，你可以从以下地址（https://github.com/cbrownley/foundations-for-analytics-with-python/blob/master/csv/supplier_data.csv）下载这个文件，步骤如下。

(1) 打开一个新的电子表格，向其中加入数据，如图 2-1 所示。



	A	B	C	D	E	F	G
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date		
2	Supplier X	001-1001	2341	$500.00	1/20/2014		
3	Supplier X	001-1001	2341	$500.00	1/20/2014		
4	Supplier X	001-1001	5467	$750.00	1/20/2014		
5	Supplier X	001-1001	5467	$750.00	1/20/2014		
6	Supplier Y	50-9501	7009	$250.00	1/30/2014		
7	Supplier Y	50-9501	7009	$250.00	1/30/2014		
8	Supplier Y	50-9505	6650	$125.00	2/3/2014		
9	Supplier Y	50-9505	6650	$125.00	2/3/2014		
10	Supplier Z	920-4803	3321	$615.00	2/3/2014		
11	Supplier Z	920-4804	3321	$615.00	2/10/2014		
12	Supplier Z	920-4805	3321	$615.00	2/17/2014		
13	Supplier Z	920-4806	3321	$615.00	2/24/2014		
14							

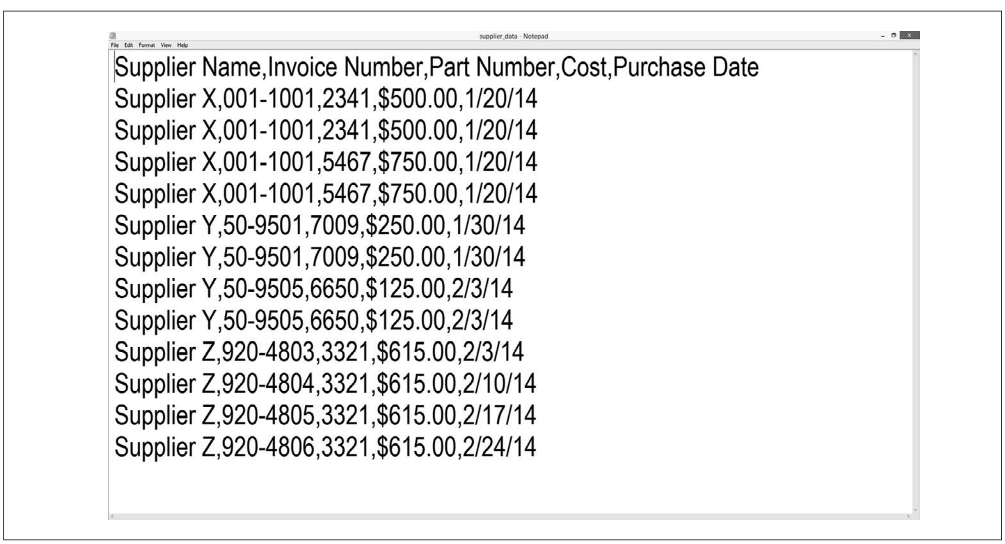
图 2-1：向 supplier_data.csv 文件中添加数据

(2) 将文件保存在桌面上，文件名为 supplier_data.csv。

要确认 supplier_data.csv 确实是纯文本文件。

(1) 将所有打开的窗口最小化，在桌面上找到 supplier_data.csv。
(2) 在文件上点击鼠标右键。
(3) 选择“Open with”，然后选择一个文本编辑器，如 Notepad、Notepad++ 或 Sublime Text。

当你在文本编辑器中打开这个文件时，它看上去应该如图 2-2 所示。



```
Supplier Name,Invoice Number,Part Number,Cost,Purchase Date
Supplier X,001-1001,2341,$500.00,1/20/14
Supplier X,001-1001,2341,$500.00,1/20/14
Supplier X,001-1001,5467,$750.00,1/20/14
Supplier X,001-1001,5467,$750.00,1/20/14
Supplier Y,50-9501,7009,$250.00,1/30/14
Supplier Y,50-9501,7009,$250.00,1/30/14
Supplier Y,50-9505,6650,$125.00,2/3/14
Supplier Y,50-9505,6650,$125.00,2/3/14
Supplier Z,920-4803,3321,$615.00,2/3/14
Supplier Z,920-4804,3321,$615.00,2/10/14
Supplier Z,920-4805,3321,$615.00,2/17/14
Supplier Z,920-4806,3321,$615.00,2/24/14
```

图 2-2：Notepad 中的 supplier_data.csv 文件

正如你所看到的，这个文件是一个简单的纯文本文件。每行包含 5 个由逗号分隔的值。对这种文件的另一种理解是由逗号划定了 Excel 电子表格中的 5 列。现在你可以关闭这个文件了。

2.1　基础Python与pandas

前言中曾提到过，本章的每一小节都提供两种版本的代码来完成具体的数据处理任务。每小节中的第一种代码版本展示了如何使用基础 Python 来完成任务。第二种版本展示了如何使用 pandas 来完成任务。你会看到，使用 pandas 完成任务相对来说更容易，需要的代码更少。所以，如果你已经理解了 pandas 简化了的编程概念和操作，只是要简单完成任务的话，pandas 版的代码就非常有用。但是，每个小节都会先介绍基础 Python 版本的代码，以使你学会如何使用通用的编程概念和操作来完成任务。通过介绍两种代码版本，希望可以给你如下选择：一是使用 pandas 快速完成任务；二是学习通用的编程技能，并在提高编码能力的基础上获得解决问题的能力。对于 pandas 版本的代码，本章的解释不会像基础 Python 版那样详细，你可以将这里的示例代码当作使用 pandas 完成任务的指导手册来使用。当你完成了本书中的练习之后，如果想成为 pandas 专家，建议你继续学习 Wes McKinney 的著作《利用 Python 进行数据分析》。

2.1.1　读写CSV文件（第1部分）

1. 基础Python，不使用csv模块

现在开始学习如何使用基础 Python 代码来读写和处理 CSV 文件（不使用内置的 `csv` 模块）。先看看下面的示例代码，然后当你使用 `csv` 模块时，就会知道代码在幕后都做了些什么。

要处理 CSV 文件，先新建一个 Python 脚本，名为 1csv_read_with_simple_parsing_and_write.py。

在 Spyder 或一个文本编辑器中输入下列代码：

```
#!/usr/bin/env python3
import sys

input_file = sys.argv[1]
output_file = sys.argv[2]

with open(input_file, 'r', newline='') as filereader:
    with open(output_file, 'w', newline='') as filewriter:
        header = filereader.readline()
        header = header.strip()
        header_list = header.split(',')
        print(header_list)
        filewriter.write(','.join(map(str,header_list))+'\n')
        for row in filereader:
            row = row.strip()
            row_list = row.split(',')
            print(row_list)
            filewriter.write(','.join(map(str,row_list))+'\n')
```

在桌面上，将程序保存为 1csv_read_with_simple_parsing_and_write.py。

图 2-3、图 2-4 和图 2-5 分别展示了使用 Anaconda Spyder、Notepad++（Windows）和 TextWrangler（macOS）编写脚本的界面。

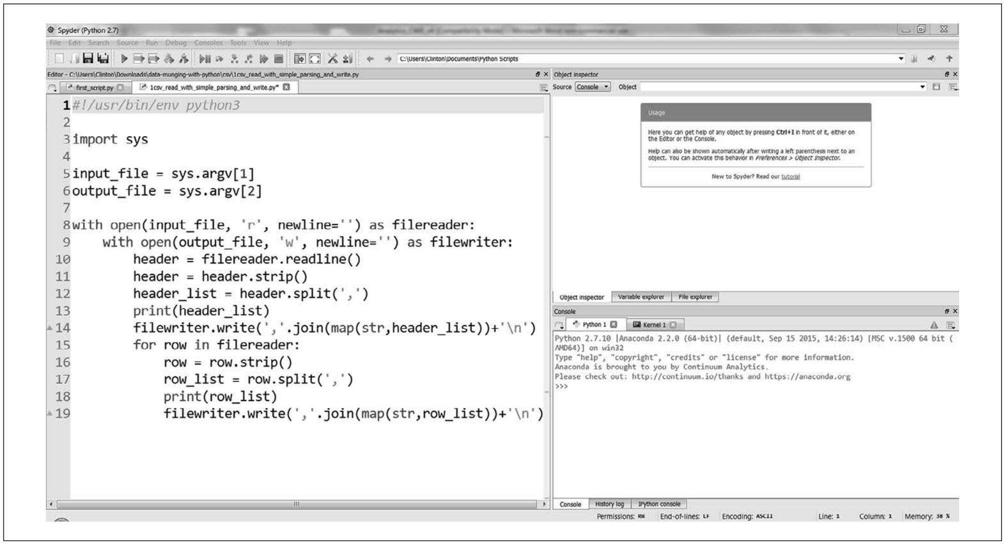


图 2-3：Anaconda Spyder 中的 Python 脚本 1csv_read_with_simple_parsing_and_write.py

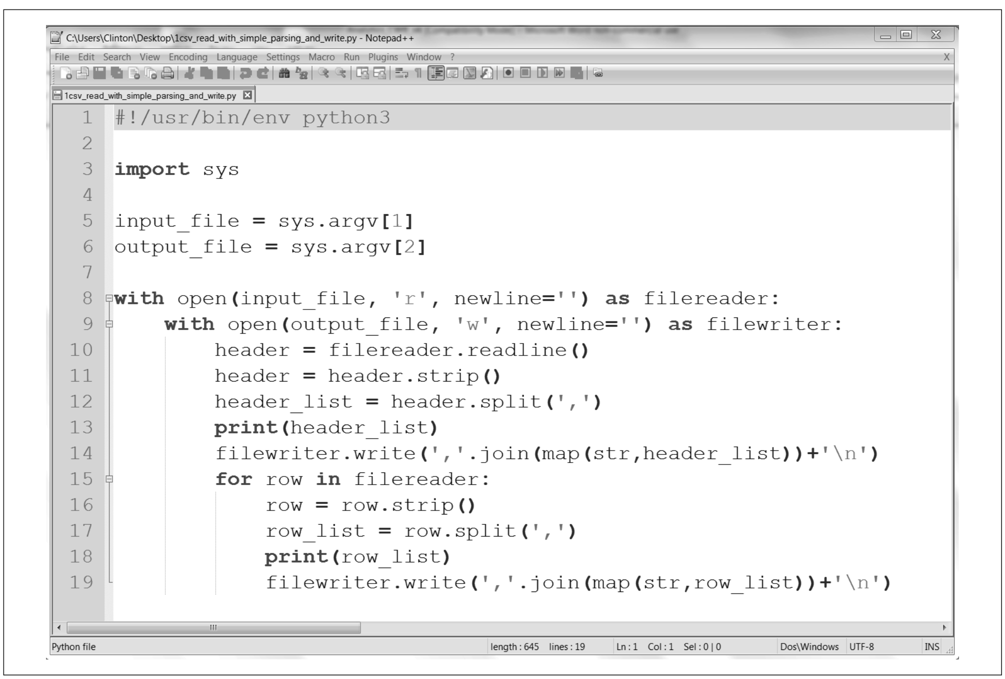


图 2-4：Notepad++（Windows）中的 Python 脚本 1csv_read_with_simple_parsing_and_write.py

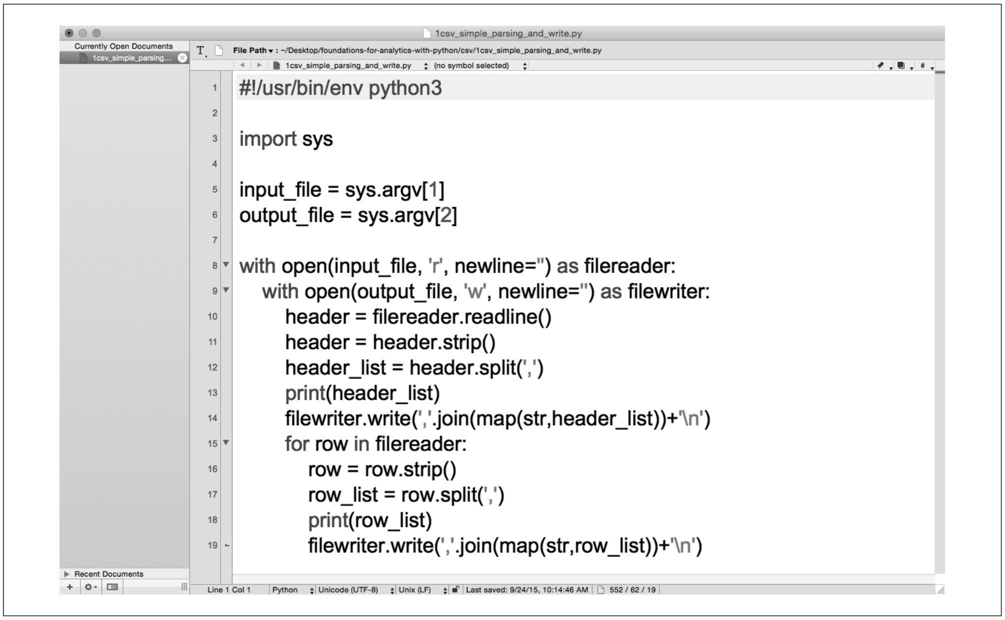


图 2-5：TextWrangler（macOS）中的 Python 脚本 1csv_read_with_simple_parsing_and_write.py

在运行脚本并查看输出之前，先研究一下脚本中的代码想做些什么。这里将按照顺序依次讨论每个代码块（下面提到的行编号指的是屏幕截图中的行编号）。

```
#!/usr/bin/env python3
import sys
```

正如第 1 章中讨论过的，第 1 行是注释行，可以使脚本在不同的操作系统之间具有可移植性。第 3 行代码导入 Python 内置的 `sys` 模块，可以使你在命令行窗口中向脚本发送附加的输入。

```
input_file = sys.argv[1]
output_file = sys.argv[2]
```

第 5 和 6 行代码使用 `sys` 模块的 `argv` 参数，它是一个传递给 Python 脚本的命令行参数列表，也就是当你运行脚本时在命令行中输入的内容。下面给出了一个在 Windows 系统中使用命令行参数读取 CSV 格式的输入文件和写入 CSV 格式的输出文件的例子：

```
python script_name.py "C:\path\to\input_file.csv" "C:\path\to\output_file.csv"
```

第一个词 `python` 告诉计算机使用 Python 程序来处理其余的命令行参数。Python 收集其余的参数，放入 `argv` 这个特殊的列表中。列表中的第一个元素 `argv[0]` 用作脚本名称，所以 `argv[0]` 表示 script_name.py。下一个命令行参数是 `"C:\path\to\input_file.csv"`，即 CSV 输入文件的路径和文件名。Python 将这个参数保存在 `argv[1]` 中，所以脚本第 5 行代码将这个值赋给变量 `input_file`。最后一个命令行参数是 `"C:\path\to\output_file.csv"`，即 CSV 输出文件的路径和文件名。Python 将这个参数保存在 `argv[2]` 中，第 6 行代码把这个值赋给了变量 `output_file`。

```
with open(input_file, 'r', newline='') as filereader:
with open(output_file, 'w', newline='') as filewriter:
```

第 8 行代码是一个 with 语句，将 input_file 打开为一个文件对象 filereader。'r' 表示只读模式，说明打开 input_file 是为了读取数据。第 9 行代码是另一个 with 语句，将 output_file 打开为一个文件对象 filewriter。'w' 表示可写模式，说明打开 output_file 是为了写入数据。正如 1.5.3 节中介绍的那样，with 语句非常有用，因为它可以在语句结束时自动关闭文件对象。

```
header = filereader.readline()
header = header.strip()
header_list = header.split(',')
```

第 10 行代码使用文件对象的 readline 方法读取输入文件中的第一行数据，在本例中，第一行是标题行，读入后将其作为字符串并赋给名为 header 的变量。第 11 行代码使用 string 模块中的 strip 函数去掉 header 中字符串两端的空格、制表符和换行符，并将处理过的字符串重新赋给 header。第 12 行代码使用 string 模块的 split 函数将字符串用逗号拆分成列表，列表中的每个值都是一个列标题，最后将列表赋给变量 header_list。

```
print(header_list)
filewriter.write(','.join(map(str,header_list))+'\n')
```

第 13 行代码是一个 print 语句，将 header_list 中的值（也就是列标题）打印到屏幕上。

第 14 行代码使用 filewriter 对象的 write 方法将 header_list 中的每个值写入输出文件。因为这行代码比较复杂，所以需要仔细说明一下。map 函数将 str 函数应用于 header_list 中的每个元素，确保每个元素都是字符串。然后，join 函数在 header_list 中的每个值之间插入一个逗号，将这个列表转换为一个字符串。在此之后，在这个字符串最后添加一个换行符。最后，filewriter 对象将这个字符串写入输出文件，作为输出文件的第一行。

```
for row in filereader:
    row = row.strip()
    row_list = row.split(',')
    print(row_list)
    filewriter.write(','.join(map(str,row_list))+'\n')
```

第 15 行代码创建了一个 for 循环，在输入文件剩余的各行中迭代。第 16 行代码使用 strip 函数除去每行字符串两端的空格、制表符和换行符，然后将处理过的字符串重新赋给变量 row。第 17 行使代码用 split 函数用逗号将字符串拆分成一个列表，列表中的每个值都是这行中某一列的值，然后，将列表赋给变量 row_list。第 18 行代码将 row_list 中的值打印到屏幕上。第 19 行代码将这些值写入输出文件。

脚本对输入文件中的每一行数据都执行第 16~19 行代码，因为这 4 行代码在第 15 行代码中的 for 循环下面是缩进的。

你可以在命令行窗口或终端窗口中通过运行脚本做一下测试。如下所示。

- 命令行窗口（Windows）

  (1) 打开一个命令行窗口。

(2) 切换到桌面（你存放 Python 脚本的地方）。

要完成这个操作，输入以下命令，然后按回车键：

```
cd "C:\Users\[Your Name]\Desktop"
```

(3) 运行 Python 脚本。

要完成这个操作，输入以下命令，然后按回车键：

```
python 1csv_simple_parsing_and_write.py supplier_data.csv\
output_files\1output.csv
```

- 终端窗口（macOS）

(1) 打开一个终端窗口。

(2) 切换到桌面（你存放 Python 脚本的地方）。

要完成这个操作，输入以下命令，然后按回车键：

```
cd /Users/[Your Name]/Desktop
```

(3) 为 Python 脚本添加可执行权限。

要完成这个操作，输入以下命令，然后按回车键：

```
chmod +x 1csv_simple_parsing_and_write.py
```

(4) 运行 Python 脚本。

要完成这个操作，输入以下命令，然后按回车键：

```
./1csv_simple_parsing_and_write.py supplier_data.csv\
output_files/1output.csv
```

如图 2-6 所示，你会看到输出被打印到命令行窗口或终端窗口中。

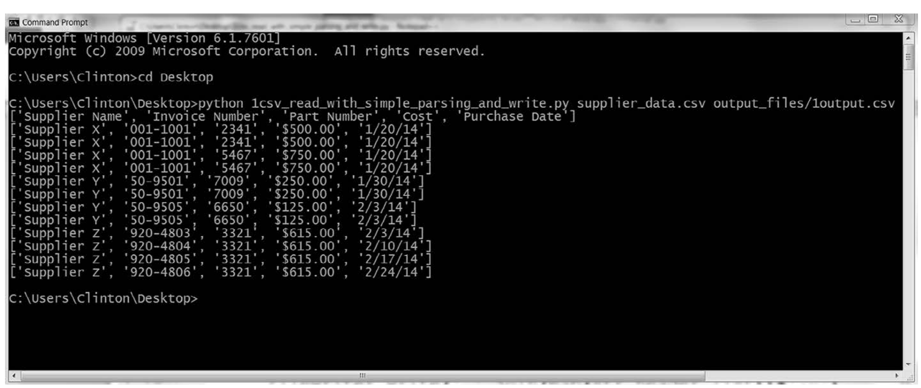


图 2-6：运行 Python 脚本 1csv_read_with_simple_parsing_and_write.py 的输出结果

输入文件中的所有行都被打印到了屏幕上，也被写入了输出文件。在多数情况下，你不需要将输入文件中的所有数据重新写到输出文件中，因为输入文件中就有所有的数据。但是

这个例子仍然是非常有用的，因为你可以参考例子中的代码，将 filewriter.write 语句嵌入到带有判断条件的业务逻辑中，确保你只将需要的某些行写入输出文件。

2. pandas

要使用 pandas 处理 CSV 文件，在文本编辑器中输入下列代码，并将文件保存为 pandas_parsing_and_write.py（这个脚本读取 CSV 文件，在屏幕上打印文件内容，并将内容写入一个输出文件）：

```
#!/usr/bin/env python3
import sys
import pandas as pd
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
print(data_frame)
data_frame.to_csv(output_file, index=False)
```

要运行这个脚本，在命令行中输入以下命令，命令在不同的操作系统中会有些差别。

- Windows 操作系统

```
python pandas_parsing_and_write.py supplier_data.csv\
output_files\pandas_output.csv
```

- macOS 操作系统

```
chmod +x pandas_parsing_and_write.py
./pandas_parsing_and_write .py supplier_data.csv\
output_files/pandas_output.csv
```


你会注意到在 pandas 版的脚本中，创建了一个变量 data_frame。同列表、字典与元组相似，数据框也是存储数据的一种方式。数据框中保留了“表格”这种数据组织方式，不需要使用列表套列表的方式来分析数据。数据框包含在 pandas 包中，如果你不在脚本中导入 pandas，就不能使用数据框。将变量命名为 data_frame，就像使用变量名 list 一样，在学习阶段，这样做是可以的，但是以后，你应该使用更有描述性的变量名。

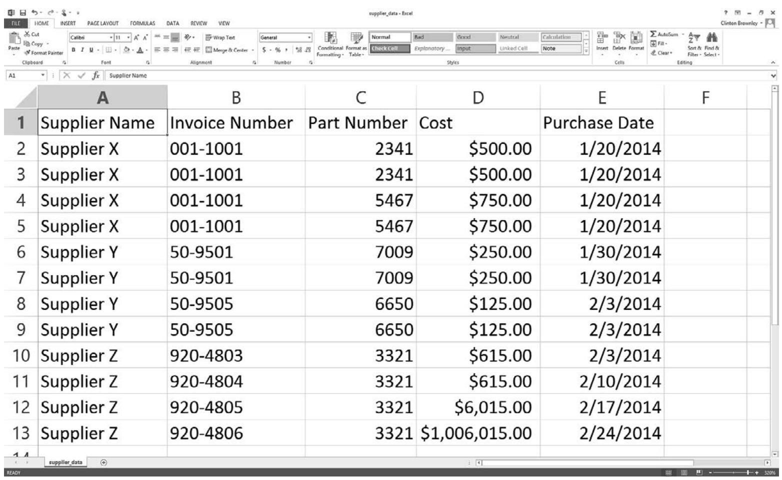
脏数据

现实世界中，数据通常是“脏”的。有些值会因为某些原因而缺失，手工输入或传感器出错都可以造成数据错误。某些情况下，人们会故意记下错误的数据，因为只能这样做。我曾经见过在餐厅收据中，将乐啤露记为“可乐（加奶酪）”，因为结账系统中没有“乐啤露”这个选项，所以使用系统的店员就加入了这个订单选项，并告知了订餐员和打饮料的服务员。但是这样一来，负责跟踪库存和订货的管理人员就有一大堆奇怪的数据需要核实了。

在电子表格数据中，你也会遇到这样的问题，并想出解决的办法。在练习各章中的示例代码时，也要注意这种情况。请记住每个人都会遇到“脏”数据的问题，这是数据分析工作中最令人头疼也是最令人兴奋的部分，通常也是工作量最大的部分，这是必须要做的工作！

2.1.2 基本字符串分析是如何失败的

基本的 CSV 分析失败的一个原因是列中包含额外的逗号。打开 supplier_data.csv，将 Cost 列中的最后两个成本数量分别改为 $6,015.00 和 $1,006,015.00。做完这两个修改之后，输入文件应如图 2-7 所示。



	A	B	C	D	E
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date
2	Supplier X	001-1001	2341	$500.00	1/20/2014
3	Supplier X	001-1001	2341	$500.00	1/20/2014
4	Supplier X	001-1001	5467	$750.00	1/20/2014
5	Supplier X	001-1001	5467	$750.00	1/20/2014
6	Supplier Y	50-9501	7009	$250.00	1/30/2014
7	Supplier Y	50-9501	7009	$250.00	1/30/2014
8	Supplier Y	50-9505	6650	$125.00	2/3/2014
9	Supplier Y	50-9505	6650	$125.00	2/3/2014
10	Supplier Z	920-4803	3321	$615.00	2/3/2014
11	Supplier Z	920-4804	3321	$615.00	2/10/2014
12	Supplier Z	920-4805	3321	$6,015.00	2/17/2014
13	Supplier Z	920-4806	3321	$1,006,015.00	2/24/2014

图 2-7：修改后的输入文件（supplier_data.csv）

修改了输入文件之后，要看看你的简单的分析脚本如何失败，需要在修改后的新输入文件上重新运行脚本。保存修改后的文件，然后按向上箭头键，找到之前运行过的命令，或者重新输入以下命令，然后按回车键：

```
python 1csv_simple_parsing_and_write.py supplier_data.csv\
output_files\1output.csv
```

你会看到输出被打印到屏幕上，如图 2-8 所示。

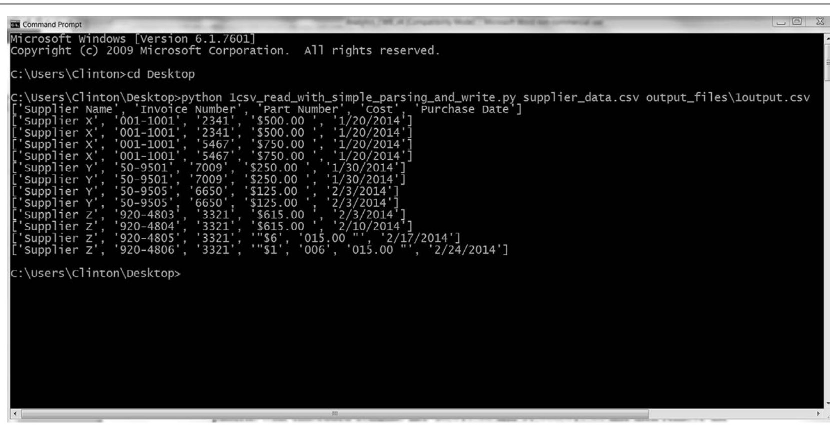


图 2-8：在修改后的 supplier_data.csv 上运行脚本

你可以看到，这里的脚本是按照行中的逗号分析每行数据的。此脚本对标题行和前 10 个数据行的处理都是正确的，因为它们没有嵌入到数据中的逗号。但是，脚本错误地拆分了最后两行，因为数据中有逗号。

有许多方法可以改进这个脚本中的代码，处理包含逗号的数值。例如，可以使用正则表达式来搜索带有嵌入逗号的模式，就像 $6,015.00 和 $1,006,015.00，然后删除这些值中的逗号，再使用余下的逗号来拆分行。但是，为了不使脚本复杂化，可以使用 Python 内置的 `csv` 模块，设计这个模块的目的就是为了方便灵活地处理复杂的 CSV 文件。

2.1.3 读写CSV文件（第2部分）

基础Python，使用csv模块

使用 Python 内置的 `csv` 模块处理 CSV 文件的一个优点是，这个模块就是被设计用于正确处理数据值中的嵌入逗号和其他复杂模式的。它可以识别出这些模式并正确地分析数据，所以你不需要仅仅为了正确处理数据而花费时间来设计正则表达式和条件逻辑，可以将节省的时间用来管理数据、执行计算和写入输出。

接下来导入 Python 内置的 `csv` 模块并用它来处理包含数值 $6,015.00 和 $1,006,015.00 的输入文件。你将学会如何使用 `csv` 模块，并理解它是如何处理数据中的逗号的。

在文本编辑器中输入以下代码，并将文件保存为 2csv_reader_parsing_and_write.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import sys
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 with open(input_file, 'r', newline='') as csv_in_file:
 7     with open(output_file, 'w', newline='') as csv_out_file:
 8          filereader = csv.reader(csv_in_file, delimiter=',')
 9          filewriter = csv.writer(csv_out_file, delimiter=',')
10          for row_list in filereader:
11               print(row_list)
12               filewriter.writerow(row_list)
```

你可以看到，上面大部分代码与前一个脚本中的代码非常相似。所以，这里只讨论那些有明显区别的代码。

第 2 行代码导入 `csv` 文件，以便可以使用其中的函数来分析输入文件，写入输出文件。

第 8 行代码，就是在第二个 `with` 语句下面的那行代码，使用 `csv` 模块中的 `reader` 函数创建了一个文件读取对象，名为 `filereader`，可以使用这个对象来读取输入文件中的行。同样，第 9 行代码使用 `csv` 模块的 `writer` 函数创建了一个文件写入对象，名为 `filewriter`，可以使用这个对象将数据写入输出文件。这些函数中的第二个参数（就是 `delimiter=','`）是默认分隔符，所以如果你的输入文件和输出文件都是用逗号分隔的，就不需要指定这个参数。这里指定了这个分隔符参数，是为了防备你处理的输入文件或要写入的输出文件具有不同的分隔符，例如，分号（`;`）或制表符（`\t`）。

第 12 行代码使用 `filewriter` 对象的 `writerow` 函数来将每行中的列表值写入输出文件。

假设输入文件和 Python 脚本都保存在你的桌面上，你也没有在命令行或终端行窗口中改变目录，在命令行中输入以下命令，然后按回车键运行脚本（如果你使用 Mac，需要对新的脚本先运行 chmod 命令，使它成为可执行的）：

```
python 2csv_reader_parsing_and_write.py supplier_data.csv\
output_files\2output.csv
```

你可以看到输出被打印到屏幕上，如图 2-9 所示。

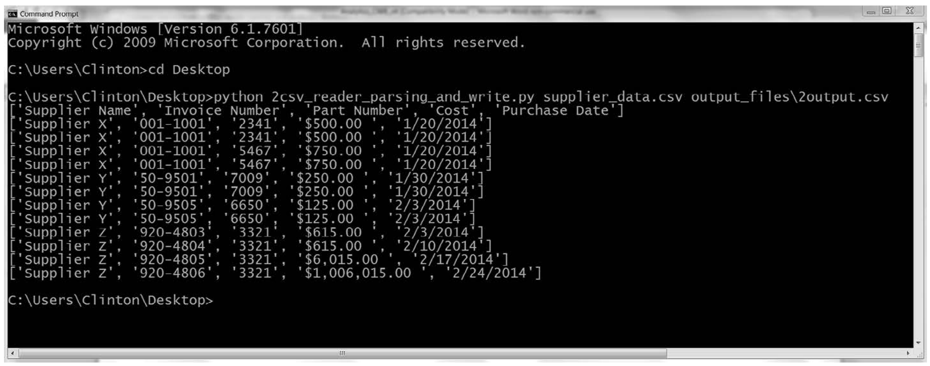


图 2-9：运行 Python 脚本得到的输出

输入文件中的所有行都被打印到了屏幕上，同时被写入到输出文件。你可以看到，Python 内置的 csv 模块处理了嵌入数据的逗号问题，正确地将每一行拆分成了 5 个值。

我们知道了如何使用 csv 模块来读取、处理和写入 CSV 文件，下面开始学习如何筛选出特定的行以及如何选择特定的列，以便可以有效地抽取出需要的数据。

2.2　筛选特定的行

有些时候，你并不需要文件中所有的数据。例如，你可能只需要一个包含特定词或数字的行的子集，或者是与某个具体日期关联的行的子集。在这些情况下，可以用 Python 筛选出特定的行来使用。

你应该很熟悉如何在 Excel 中手动筛选行，但是本章的重点在于提高你的能力，使你既能处理因为体积太大以致 Excel 不能打开的 CSV 文件，又能处理多个 CSV 文件。因为要通过手动处理这些文件，时间花费太多了。

下面各小节演示了在输入文件中筛选出特定行的 3 种方法：

- 行中的值满足某个条件
- 行中的值属于某个集合
- 行中的值匹配于某个模式（正则表达式）

你会发现这些小节中的代码在结构上是一致的。接下来会详细解释这种通用结构，使你可以轻松地修改代码来满足自己的业务规则。

在下面的 3 个小节中，请注意以下结构，从而来理解如何从输入文件中筛选出特定的行：

```
for row in filereader:
    ***if value in row meets some business rule or set of rules:***
        do something
    else:
        do something else
```

这段伪代码展示了用来在输入文件中筛选出特定行的通用代码结构。在下面的小节中，会修改封装在 *** 之间的代码，以使脚本能够满足具体业务规则，抽取出你需要的数据。

2.2.1　行中的值满足某个条件

1. 基础Python

有些时候，当行中的值满足一个具体条件时，才需要保留这些行。例如，你可能会希望在数据集中保留那些成本高于某个具体阈值的行，或者希望保留所有购买日期在一个具体日期之前的行。在这种情况下，你可以检验行中的值是否满足具体的条件，然后筛选出满足条件的行。

下面的示例演示了检验行值是否满足两个具体条件的方法，并将满足条件的行的子集写入一个输出文件。在这个示例中，保留供应商名字为 `Supplier Z` 或成本大于 $600.00 的行，并将结果写入输出文件。要筛选出满足这些条件的行的子集，在文本编辑器中输入以下代码，将文件保存为 3csv_reader_value_meets_condition.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import sys
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 with open(input_file, 'r', newline='') as csv_in_file:
 7     with open(output_file, 'w', newline='') as csv_out_file:
 8          filereader = csv.reader(csv_in_file)
 9          filewriter = csv.writer(csv_out_file)
10          header = next(filereader)
11          filewriter.writerow(header)
12          for row_list in filereader:
13               supplier = str(row_list[0]).strip()
14               cost = str(row_list[3]).strip('$').replace(',', '')
15               if supplier == 'Supplier Z' or float(cost) > 600.0:
16                    filewriter.writerow(row_list)
```

第 10 行代码使用 `csv` 模块的 `next` 函数读出输入文件的第一行，赋给名为 `header` 的列表变量。第 11 行代码将标题行写入输出文件。

第 13 行代码取出每行数据中的供应商名字，并赋给名为 `supplier` 的变量。这行代码使用列表索引取出每行数据的第一个值 `row[0]`，然后使用 `str` 函数将其转换为一个字符串。在此之后，使用 `strip` 函数删除字符串两端的空格、制表符和换行符。最后，将处理好的字符串赋给变量 `supplier`。

第 14 行代码取出每行数据中的成本，并赋给名为 `cost` 的变量。这行代码使用列表索引取

出每行数据的第四个值 row[3]，然后使用 str 函数将其转换为一个字符串。在此之后，使用 strip 函数从字符串中删除美元符号。接着使用 replace 函数从字符串中删除逗号。最后，将处理好的字符串赋给变量 cost。

第 15 行代码创建了一个 if 语句，来检验每行中的这两个值是否满足条件。具体来说，这里想筛选出供应商名字为 Supplier Z 或者成本大于 $600.00 的那些行。if 和 or 之间的第一个条件检验变量 supplier 中的值是否为 Supplier Z。or 和冒号之间的第二个条件检验变量 cost 中的值在被转换为浮点数之后，是否大于 600.0。

第 16 行代码使用 filewriter 的 writerow 函数将满足条件的行写入输出文件。

要运行这个脚本，输入以下命令，然后按回车键：

```
python 3csv_reader_value_meets_condition.py supplier_data.csv\
output_files\3output.csv
```

在屏幕上你不会看到任何输出，但可以打开输出文件 3output.csv 看一下结果。检查一下，确保结果正确，然后可以修改一下代码，设定不同的供应商或成本阈值，试着筛选一下其他数据。

2. pandas

pandas 提供了一个 loc 函数，可以同时选择特定的行与列。你需要在逗号前面设定行筛选条件，在逗号后面设定列筛选条件。下面的 loc 函数中的条件设置为：Supplier Name 列中姓名包含 Z，或者 Cost 列中的值大于 600.0，并且需要所有的列。在文本编辑器中输入以下代码，将文件保存为 pandas_value_meets_condition.py（这个脚本使用 pandas 来分析 CSV 文件，并将满足条件的行写入输出文件）。

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame['Cost'] = data_frame['Cost'].str.strip('$').astype(float)
data_frame_value_meets_condition = data_frame.loc[(data_frame['Supplier Name']\
.str.contains('Z')) | (data_frame['Cost'] > 600.0), :]
data_frame_value_meets_condition.to_csv(output_file, index=False)
```

在命令行中运行脚本，并给出数据源文件和输出文件。

```
python pandas_value_meets_condition.py supplier_data.csv\
output_files\pandas_output.csv
```

在屏幕上你不会看到任何输出，但可以打开输出文件 pandas_output.csv 看一下结果。试试修改一下 loc 函数中的参数，选择出另外一些数据。

2.2.2　行中的值属于某个集合

1. 基础Python

有些时候，当行中的值属于某个集合时，才需要保留这些行。例如，你可能会希望在

数据集中保留那些供应商名字属于集合 {Supplier X, Supplier Y} 的行（这里的花括号表示集合，不是 Python 中的字典），或者希望保留所有购买日期属于集合 {'1/20/14', '1/30/14'} 的行。在这种情况下，你可以检验行中的值是否属于某个集合，然后筛选出具有属于该集合的值的行。

下面的示例演示了检验行值是否是集合成员的方法，并将具有集合中的值的行写入到输出文件。在这个示例中，是要保留那些购买日期属于集合 {'1/20/14', '1/30/14'} 的行，并将结果写入输出文件。要筛选出值属于这个集合的行的子集，在文本编辑器中输入以下代码，并将文件保存为 4csv_reader_value_in_set.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import sys
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 important_dates = ['1/20/14', '1/30/14']
 7 with open(input_file, 'r', newline='') as csv_in_file:
 8     with open(output_file, 'w', newline='') as csv_out_file:
 9          filereader = csv.reader(csv_in_file)
10          filewriter = csv.writer(csv_out_file)
11          header = next(filereader)
12          filewriter.writerow(header)
13          for row_list in filereader:
14               a_date = row_list[4]
15               if a_date in important_dates:
16                    filewriter.writerow(row_list)
```

第 6 行代码创建了一个名为 important_dates 的列表变量，其中包含两个特定日期，这个变量就是你的集合。创建包含特定值的变量，然后在代码中引用变量，这种编写代码的方式非常有用。使用了这种方式，如果变量值发生了变化，你只需在一个地方修改代码（就是定义变量的地方），变量值的变化就会反映到代码中所有引用该变量的地方。

第 14 行代码取出每一行的购买日期，并将其赋给变量 a_date。从行列表的索引值 row[4] 可知，购买日期在第 5 列。

第 15 行代码创建了一个 if 语句来检验 a_date 变量中的购买日期是否属于 important_dates 这个集合。如果变量值在集合中，下一行代码就将这一行写入输出文件。

在命令行中运行下面脚本：

```
python 4csv_reader_value_in_set.py supplier_data.csv output_files/4output.csv
```

你可以打开输出文件 4output.csv 来检查结果。

2. pandas

当行中的值属于某个集合时，如何使用 pandas 筛选出这些行呢？在文本编辑器中输入以下代码，然后将文件保存为 pandas_value_in_set.py（这个脚本分析 CSV 文件，并将值属于某个集合的行写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
```

```
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
important_dates = ['1/20/14', '1/30/14']
data_frame_value_in_set = data_frame.loc[data_frame['Purchase Date'].\
isin(important_dates), :]
data_frame_value_in_set.to_csv(output_file, index=False)
```

这里最重要的新命令就是简洁的 `isin`。

和以前一样，在命令行中运行脚本，并给出源数据文件名和输出文件名：

```
python pandas_value_in_set.py supplier_data.csv output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 来检查结果。

2.2.3　行中的值匹配于某个模式/正则表达式

1. 基础Python

有些时候，当行中的值匹配了或包含了一个特定模式（也就是正则表达式）时，才需要保留这些行。例如，你可能会希望在数据集中保留所有发票编号开始于“001-”的行，或者希望保留所有供应商名字中包含“Y”的行。在这种情况下，你可以检验行中的值是否匹配或包含某种模式，然后筛选出匹配了或包含了该模式的行。

下面的示例演示了如何检验某个值是否匹配特定的模式，并将具有这种值的行写入输出文件。在这个示例中，保留发票编号由“001-”开头的行，并将结果写入一个输出文件。要筛选出某个值匹配了这个模式的行，在文本编辑器中输入下列代码，然后将文件保存为 5csv_reader_value_matches_pattern.py：

```
1 #!/usr/bin/env python3
2 import csv
3 import re
4 import sys
5 input_file = sys.argv[1]
6 output_file = sys.argv[2]
7 pattern = re.compile(r'(?P<my_pattern_group>^001-.*)', re.I)
8 with open(input_file, 'r', newline='') as csv_in_file:
9     with open(output_file, 'w', newline='') as csv_out_file:
10         filereader = csv.reader(csv_in_file)
11         filewriter = csv.writer(csv_out_file)
12         header = next(filereader)
13         filewriter.writerow(header)
14         for row_list in filereader:
15             invoice_number = row_list[1]
16             if pattern.search(invoice_number):
17                 filewriter.writerow(row_list)
```

第 3 行代码导入正则表达式（`re`）模块，这样就可以使用 `re` 模块中的函数了。

第 7 行代码使用 `re` 模块的 `compile` 函数创建一个名为 `pattern` 的正则表达式变量。如果你学习了第 1 章，那么应该很熟悉这个函数。`r` 表示将单引号之间的模式当作原始字符

串来处理。

元字符 ?P<my_pattern_group> 捕获了名为 <my_pattern_group> 的组中匹配了的子字符串，以便在需要时将它们打印到屏幕或写入文件。

这里要搜索的实际模式是 ^001-.*。插入符号（^）是一个特殊符号，表示只在字符串开头搜索模式。所以，字符串需要以“001-”开头。句点 . 可以匹配任何字符，除了换行符。所以除换行符之外的任何字符都可以跟在“001-”后面。最后，* 表示重复前面的字符 0 次或更多次。.* 组合在一起用来表示除换行符之外的任意字符可以在“001-”后面出现任意次。更通俗的说法是：字符串在“-”后面可以包含任意值，只要字符串开始于“001-”，就会匹配正则表达式。

最后，参数 re.I 告诉正则表达式进行大小写敏感的匹配。此参数在这个示例中不是太重要，因为模式是数值型的，但是它说明了在模式中包含字符并且需要进行大小写敏感的匹配时，应该如何设置参数。

第 15 行代码使用列表索引从行中取出发票编号，并赋给变量 invoice_number。在下一行中，将在这个变量中寻找模式。

第 16 行代码使用 re 模块的 search 函数在 invoice_number 的值中寻找模式。如果模式出现在 invoice_number 值中，第 17 行代码就将这行写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 5csv_reader_value_matches_pattern.py supplier_data.csv\
output_files\5output.csv
```

你可以打开输出文件 5output.csv 来查看结果。

2. pandas

要使用 pandas 筛选出匹配于某个模式的行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_matches_pattern.py（这个脚本读取 CSV 文件，将匹配于某个模式的行打印在屏幕上，并将同样的行写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_value_matches_pattern = data_frame.loc[data_frame['Invoice Number'].\
str.startswith("001-"), :]
data_frame_value_matches_pattern.to_csv(output_file, index=False)
```

使用 pandas 时，可以使用 startwith 函数来搜索数据，不用再使用笨重冗长的正则表达式了。要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_value_matches_pattern.py supplier_data.csv\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

2.3 选取特定的列

有些时候，你并不需要文件中所有的列。在本节示例中，可以使用 Python 选取出你需要的列。

有两种通用方法可以在 CSV 文件中选取特定的列。下面各小节演示了这两种方法：

- 使用列索引值
- 使用列标题

2.3.1 列索引值

1. 基础Python

在 CSV 文件中选取特定列的一种方法是使用你想保留的列的索引值。当你想保留的列的索引值非常容易识别，或者在处理多个输入文件时，各个输入文件中列的位置一致（也就是不会发生改变）的时候，这种方法非常有效。例如，如果你只需要保留数据的第一列和最后一列，那么你可以使用 `row[0]` 和 `row[-1]` 来将每行的第一个值和最后一个值写入文件。

在这个示例中，你只想保留供应商姓名和成本这两列。要使用索引值选取这两列，在文本编辑器中输入下列代码，然后将文件保存为 6csv_reader_column_by_index.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import sys
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 my_columns = [0, 3]
 7 with open(input_file, 'r', newline='') as csv_in_file:
 8     with open(output_file, 'w', newline='') as csv_out_file:
 9         filereader = csv.reader(csv_in_file)
10         filewriter = csv.writer(csv_out_file)
11         for row_list in filereader:
12             row_list_output = [ ]
13             for index_value in my_columns:
14                 row_list_output.append(row_list[index_value])
15             filewriter.writerow(row_list_output)
```

第 6 行代码创建了一个列表变量 `my_columns`，其中包含了你想保留的两列的索引值。在这个示例中，这两个索引值对应着供应商姓名和成本列。再说一次，应该创建一个包含索引值的变量，然后在代码中引用这个变量。这样，如果索引值需要改变的话，你只需要在一个地方（就是定义 `my_columns` 的地方）修改即可，修改会反映到代码中所有引用 `my_columns` 的地方。

第 12~15 行代码是 `for` 循环下面缩进的部分，所以对于输入文件中的每一行都要执行这些代码。第 12 行代码创建了一个空列表变量 `row_list_output`。这个变量保存你在每行中要保留的值。第 13 行代码是一个 `for` 循环语句，在 `my_cloumns` 中的各个索引值之间进行迭代。第 14 行代码通过列表的 `append` 函数使用每行中 `my_columns` 索引位置的值为 `row_list_output` 填充元素。这 3 行代码生成了一个列表，列表中包含了每行中你要写入输出文件的值。创建列表是有用的，因为 `filewriter` 的 `writerow` 方法需要一个字符串序列或

数值序列，而列表 row_list_out 正是一个字符串序列。第 15 行代码将 row_list_output 中的值写入输出文件。

脚本会对输入文件中的每一行执行这些代码。为了确切地理解这一系列操作，下面来看看第一次外部 for 循环做了些什么。在本例中，你处理的是输入文件中的第一行（也就是标题行）。第 12 行代码创建了空列表变量 row_list_output。第 13 行代码是一个 for 循环，在 my_columns 的值之间迭代。

第一次循环时，index_value 等于 0，所以第 14 行代码中的 append 函数将 row[0]（就是供应商姓名字符串）加入 row_list_output。此后，代码回到第 13 行中的 for 循环，这一次 index_value 等于 3。因为 index_value 等于 3，所以第 14 行代码中的 append 函数将 row[3]（也就是成本字符串）加入 row_list_output。my_columns 中没有更多的值了，所以第 13 行中的 for 循环结束，代码前进到第 15 行。第 15 行代码将 row_list_output 中的列表值写入输出文件。然后，代码回到第 11 行中的外部 for 循环，开始处理输入文件中的下一行。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 6csv_reader_column_by_index.py supplier_data.csv output_files\6output.csv
```

你可以打开输出文件 6output.csv 查看一下结果。

2. pandas

要使用 pandas 根据索引值选取列，在文本编辑器中输入下列代码，然后将文件保存为 pandas_column_by_index.py（这个脚本读取 CSV 文件，将索引值为 0 和 3 的列打印到屏幕，并将同样的行写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_column_by_index = data_frame.iloc[:, [0, 3]]
data_frame_column_by_index.to_csv(output_file, index=False)
```

这里使用了 iloc 函数来根据索引位置选取列。

在命令行中运行以下脚本：

```
python pandas_column_by_index.py supplier_data.csv\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

2.3.2　列标题

1. 基础Python

在 CSV 文件中选取特定列的第二种方法是使用列标题，而不是索引位置。当你想保留的列的标题非常容易识别，或者在处理多个输入文件时，各个输入文件中列的位置会发生改变，但标题不变的时候，这种方法非常有效。

举例来说，假设你只需要保留发票号码列和购买日期列。要使用列标题选取这两列，在文本编辑器中输入下列代码，然后将文件保存为 7csv_reader_column_by_name.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import sys
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 my_columns = ['Invoice Number', 'Purchase Date']
 7 my_columns_index = []
 8 with open(input_file, 'r', newline='') as csv_in_file:
 9     with open(output_file, 'w', newline='') as csv_out_file:
10         filereader = csv.reader(csv_in_file)
11         filewriter = csv.writer(csv_out_file)
12         header = next(filereader, None)
13         for index_value in range(len(header)):
14             if header[index_value] in my_columns:
15                 my_columns_index.append(index_value)
16         filewriter.writerow(my_columns)
17         for row_list in filereader:
18             row_list_output = [ ]
19             for index_value in my_columns_index:
20             row_list_output.append(row_list[index_value])
21         filewriter.writerow(row_list_output)
```

这个示例中的代码比上一个示例要稍微长一点，但是所有代码看起来都很熟悉。此示例中有更多代码的唯一原因就是，你需要先单独处理一下标题行，识别出相应标题行对应的索引值。然后你可以使用索引值保留每行中的值，这些值和要保留的列标题具有同样的索引值。

第 6 行代码创建了一个列表变量 my_columns，其中包含了两个字符串，即要保留的两列的名字。第 7 行代码创建了一个空列表变量 my_columns_index，要使用两个保留列的索引值来填充它。

第 12 行代码在 filereader 对象上使用 next 函数从输入文件中读出第一行，并保存在列表变量 header 中。第 13 行代码初始化在列标题的索引值中迭代的 for 循环。

第 14 行代码使用 if 语句和列表索引来检验每个列标题是否在 my_columns 中。例如，第一次 for 循环时，index_value 等于 0，所以 if 语句检验 header[0]（也就是第一个列标题供应商姓名）是否在 my_columns 中。因为供应商姓名不在 my_columns 中，所以第 15 行代码不会对这个值执行。

代码返回第 13 行中的 for 循环，这一次 index_value 等于 1。然后，第 14 行代码中的 if 语句检验 header[1]（也就是第二个列标题发票号码）是否在 my_columns 中。因为发票号码在 my_columns 中，所以执行第 15 行代码，将这列的索引值加入到 my_columns_index 列表中。

然后继续 for 循环，最后将购买日期列的索引值加入 my_columns_index。一旦 for 循环结束，第 16 行代码就将 my_columns 中的两个字符串写入输出文件。

第 18~21 行代码处理输入文件中余下的数据行。第 18 行代码创建一个空列表 row_list_

output 来保存你要在每一行中保留的值。第 19 行代码中的 for 循环在 my_columns_index 中的索引值之间迭代，第 20 行代码将数据行中具有这些索引值的值加入 row_list_output。最后，第 21 行代码将 row_list_output 中的值写入输出文件。

在命令行中运行以下脚本：

```
python 7csv_reader_column_by_name.py supplier_data.csv output_files\7output.csv
```

你可以打开输出文件 7output.csv 查看一下结果。

2. pandas

要使用 pandas 根据列标题选取列，在文本编辑器中输入下列代码，然后将文件保存为 pandas_column_by_name.py（这个脚本读取 CSV 文件，将发票号码列与购买日期列打印到屏幕，并将同样的列写入输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame_column_by_name = data_frame.loc[:, ['Invoice Number', 'Purchase Date']]
data_frame_column_by_name.to_csv(output_file, index=False)
```

这里又一次使用 loc 函数来选取列，这次使用的是列标题。

运行以下脚本：

```
python pandas_column_by_name.py supplier_data.csv output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

2.4 选取连续的行

有些时候，在文件内容中，工作表头部和尾部都是你不想处理的。例如，文件头部可能是标题和作者信息，文件尾部也可能会列出来源、假设、附加说明和注意事项。在很多情况下，你不需要处理这些内容。

为了演示如何在 CSV 文件中选取连续的行，需要对输入文件做如下修改。

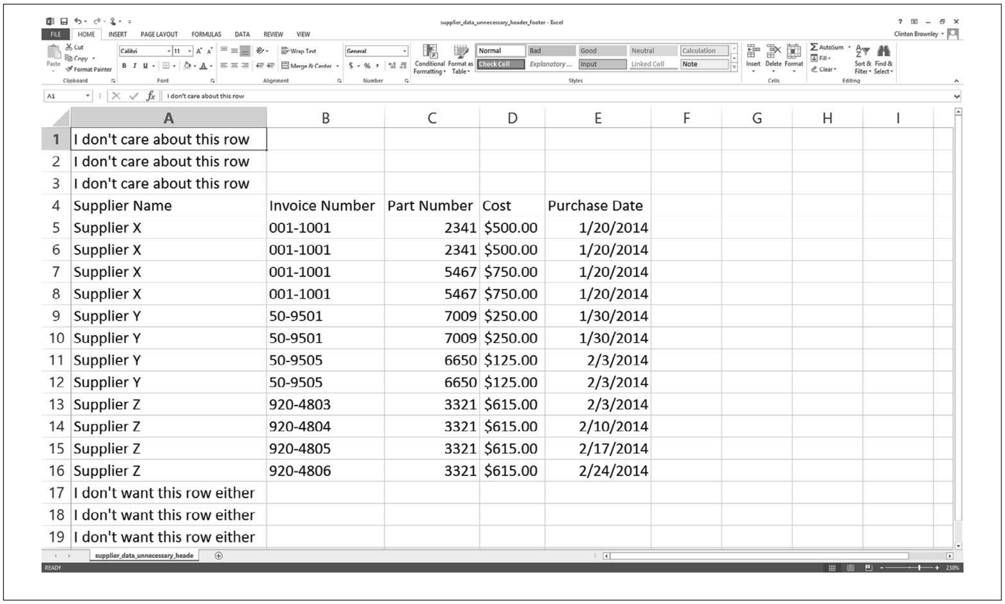
(1) 在电子表格软件中打开 supplier_data.csv。
(2) 在文件头部插入 3 行，就在列标题那行的上面。

在 A1:A3 单元格中随便写一些文字，比如“I don’t care about this line”。

(3) 在文件尾部，也就是最后一行数据下面插入 3 行。

在最后一行数据下面 A 列的 3 个单元格中随便写一些文字，比如“I don’t want this line either”。

(4) 将文件保存为 supplier_data_unnecessary_header_footer.csv。这个文件应该如图 2-10 所示。



	A	B	C	D	E
1	I don't care about this row				
2	I don't care about this row				
3	I don't care about this row				
4	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date
5	Supplier X	001-1001	2341	$500.00	1/20/2014
6	Supplier X	001-1001	2341	$500.00	1/20/2014
7	Supplier X	001-1001	5467	$750.00	1/20/2014
8	Supplier X	001-1001	5467	$750.00	1/20/2014
9	Supplier Y	50-9501	7009	$250.00	1/30/2014
10	Supplier Y	50-9501	7009	$250.00	1/30/2014
11	Supplier Y	50-9505	6650	$125.00	2/3/2014
12	Supplier Y	50-9505	6650	$125.00	2/3/2014
13	Supplier Z	920-4803	3321	$615.00	2/3/2014
14	Supplier Z	920-4804	3321	$615.00	2/10/2014
15	Supplier Z	920-4805	3321	$615.00	2/17/2014
16	Supplier Z	920-4806	3321	$615.00	2/24/2014
17	I don't want this row either				
18	I don't want this row either				
19	I don't want this row either				

图 2-10：在你需要的行上方和下方具有无关数据的 CSV 文件

现在输入文件中包含了你不需要的头部和尾部信息，修改一下 Python 脚本，使它不读取这些行。

1. 基础Python

要使用基础 Python 选取特定行，这里使用 `row_counter` 变量来跟踪行编号，以便可以识别和选取想保留的行。从前面的示例中，你已经知道了要保留 13 行数据。在下面的 `if` 代码块中，你可以看到你要写入输出文件中的行就是行索引大于等于 3 并小于等于 15 的行。

要使用基础 Python 选取这些行，在文本编辑器中输入下列代码，然后将文件保存为 11csv_reader_select_contiguous_rows.py：

```python
#!/usr/bin/env python3
import csv
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
row_counter = 0
with open(input_file, 'r', newline='') as csv_in_file:
    with open(output_file, 'w', newline='') as csv_out_file:
        filereader = csv.reader(csv_in_file)
        filewriter = csv.writer(csv_out_file)
        for row in filereader:
            if row_counter >= 3 and row_counter <= 15:
                filewriter.writerow([value.strip() for value in row])
            row_counter += 1

```

这里使用 `row_counter` 变量和一个 `if` 语句来保留需要的行，跳过那些不需要的头部和尾

部内容。对于输入文件的前 3 行，因为 row_counter 小于 3，所以不执行 if 代码块，并将 row_counter 的值增加 1。

对于输入文件的最后 3 行，row_counter 大于 15，所以也不执行 if 代码块，并将 row_counter 的值增加 1。

你要保留的行在无用的头部和尾部之间。对于这些行，row_counter 在 3 和 15 之间。if 代码块处理这些行并将它们写入输出文件。在列表生成式中使用 string 模块的 strip 函数除去每行两端的空格、制表符和换行符。

如果想看看 row_counter 变量的值和每行的内容，可以在现有的 writerow 语句上面加上一个 print 语句，比如 print(row_counter, [value.strip() for value in row])。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 11csv_reader_select_contiguous_rows.py supplier_data_unnecessary_header_\
footer.csv output_files\11output.csv
```

你可以打开输出文件 11output.csv 查看一下结果。

2. pandas

pandas 提供了 drop 函数根据行索引或列标题来丢弃行或列。在下面的脚本中，drop 函数从输入文件中丢弃前 3 行和最后 3 行（也就是行索引为 0，1，2 和 16，17，18 的那些行）。pandas 还提供了功能强大的 iloc 函数，你可以使用这个函数根据行索引选取一个单独行作为列索引。最后，使用 reindex 函数为数据框重新生成索引。

使用 pandas 可以保留列标题行和数据行，除去不需要的头部和尾部。在文本编辑器中输入下列代码，并将文件保存为 pandas_select_contiguous_rows.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file, header=None)
data_frame = data_frame.drop([0,1,2,16,17,18])
data_frame.columns = data_frame.iloc[0]
data_frame = data_frame.reindex(data_frame.index.drop(3))
data_frame.to_csv(output_file, index=False)
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_select_contiguous_rows.py supplier_data_unnecessary_header_\
footer.csv output_files\pandas_output.csv
```

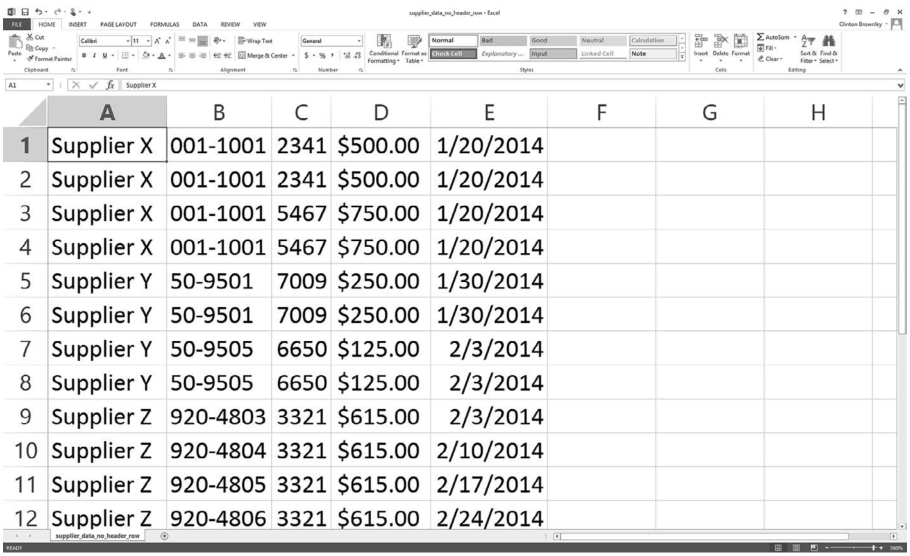
你可以打开输出文件 pandas_output.csv 查看一下结果。

2.5　添加标题行

有些时候，电子表格中没有标题行，但你确实希望所有列都有列标题。在这种情况下，可以使用脚本添加列标题。

为了演示如何使用脚本添加列标题，需要对输入文件做一下修改：

(1) 在电子表格程序中打开 supplier_data.csv。
(2) 删除文件中的第一行（即包含列标题的标题行）。
(3) 将文件保存为 supplier_data_no_header_row.csv。如图 2-11 所示。



	A	B	C	D	E	F	G	H
1	Supplier X	001-1001	2341	$500.00	1/20/2014			
2	Supplier X	001-1001	2341	$500.00	1/20/2014			
3	Supplier X	001-1001	5467	$750.00	1/20/2014			
4	Supplier X	001-1001	5467	$750.00	1/20/2014			
5	Supplier Y	50-9501	7009	$250.00	1/30/2014			
6	Supplier Y	50-9501	7009	$250.00	1/30/2014			
7	Supplier Y	50-9505	6650	$125.00	2/3/2014			
8	Supplier Y	50-9505	6650	$125.00	2/3/2014			
9	Supplier Z	920-4803	3321	$615.00	2/3/2014			
10	Supplier Z	920-4804	3321	$615.00	2/10/2014			
11	Supplier Z	920-4805	3321	$615.00	2/17/2014			
12	Supplier Z	920-4806	3321	$615.00	2/24/2014			

图 2-11：包含数据行的 CSV 文件，没有标题行

1. 基础Python

要使用基础 Python 添加列标题，在文本编辑器中输入下列代码，然后将文件保存为 12csv_reader_add_header_row.py：

```
1 #!/usr/bin/env python3
2 import csv
3 import sys
4 input_file = sys.argv[1]
5 output_file = sys.argv[2]
6 with open(input_file, 'r', newline='') as csv_in_file:
7     with open(output_file, 'w', newline='') as csv_out_file:
8          filereader = csv.reader(csv_in_file)
9          filewriter = csv.writer(csv_out_file)
10         header_list = ['Supplier Name', 'Invoice Number',\
11                        'Part Number', 'Cost', 'Purchase Date']
12         filewriter.writerow(header_list)
13         for row in filereader:
14             filewriter.writerow(row)
```

第 10 行代码创建了列表变量 header_list，其中包含了要作为列标题的 5 个字符串。第 12 行代码将这些列表值写入输出文件的第一行。同样，第 14 行代码将所有数据行写入输出

文件，放在标题行下面。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 12csv_reader_add_header_row.py supplier_data_no_header_row.csv\
output_files\12output.csv
```

你可以打开输出文件 12output.csv 查看一下结果。

2. pandas

pandas 中的 `read_csv` 函数可以直接指定输入文件不包含标题行，并可以提供一个列标题列表。要给一个没有标题行的数据集添加标题行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_add_header_row.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
header_list = ['Supplier Name', 'Invoice Number',\
               'Part Number', 'Cost', 'Purchase Date']
data_frame = pd.read_csv(input_file, header=None, names=header_list)
data_frame.to_csv(output_file, index=False)
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_add_header_row.py supplier_data_no_header_row.csv\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

2.6 读取多个CSV文件

本章到目前为止，都在演示如何处理单个 CSV 文件。有些时候，你也只需要处理一个文件。在这些情况下，上面的示例可以告诉你如何使用 Python 程序去处理。尽管只是一个文件，这个文件也可能太大，不能手工处理，因此用程序处理文件还可以减少人为犯错的概率，比如复制 / 粘贴错误和输入错误。

但是，在大多数情况下，你需要处理的文件很多，多到使用手工处理效率非常低或者根本不可行。在这种情况下，Python 会给你惊喜，因为它可以让你自动化和规模化地进行数据处理，远远超过手工处理能够达到的限度。这一小节介绍 Python 内置的 `glob` 模块，并在本章前面示例的基础上，演示如何规模化地处理 CSV 文件。

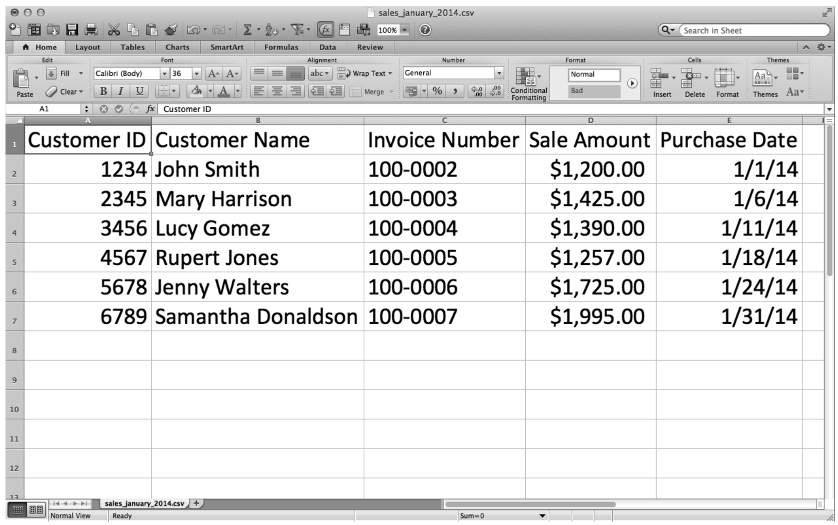
为了处理多个 CSV 文件，首先需要创建多个 CSV 文件。下面的示例中创建了 3 个 CSV 文件，但是请记住，这里介绍的技术可以扩展为处理计算机允许的任意多的文件，多到几百个，甚至更多！

- 第一个 CSV 文件

  (1) 打开一个电子表格程序。

  (2) 加入图 2-12 所示的数据。

(3) 将文件保存为 sales_january_2014.csv。



Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
1234	John Smith	100-0002	$1,200.00	1/1/14
2345	Mary Harrison	100-0003	$1,425.00	1/6/14
3456	Lucy Gomez	100-0004	$1,390.00	1/11/14
4567	Rupert Jones	100-0005	$1,257.00	1/18/14
5678	Jenny Walters	100-0006	$1,725.00	1/24/14
6789	Samantha Donaldson	100-0007	$1,995.00	1/31/14

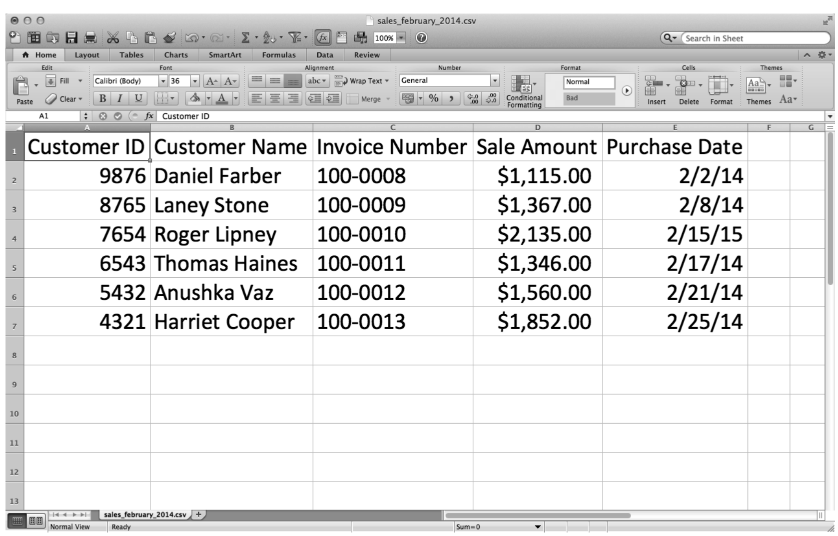
图 2-12：第一个 CSV 文件：sales_january_2014.csv

- 第二个 CSV 文件

  (1) 打开一个电子表格程序。

  (2) 加入图 2-13 所示的数据。

  (3) 将文件保存为 sales_february_2014.csv。



Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
9876	Daniel Farber	100-0008	$1,115.00	2/2/14
8765	Laney Stone	100-0009	$1,367.00	2/8/14
7654	Roger Lipney	100-0010	$2,135.00	2/15/15
6543	Thomas Haines	100-0011	$1,346.00	2/17/14
5432	Anushka Vaz	100-0012	$1,560.00	2/21/14
4321	Harriet Cooper	100-0013	$1,852.00	2/25/14

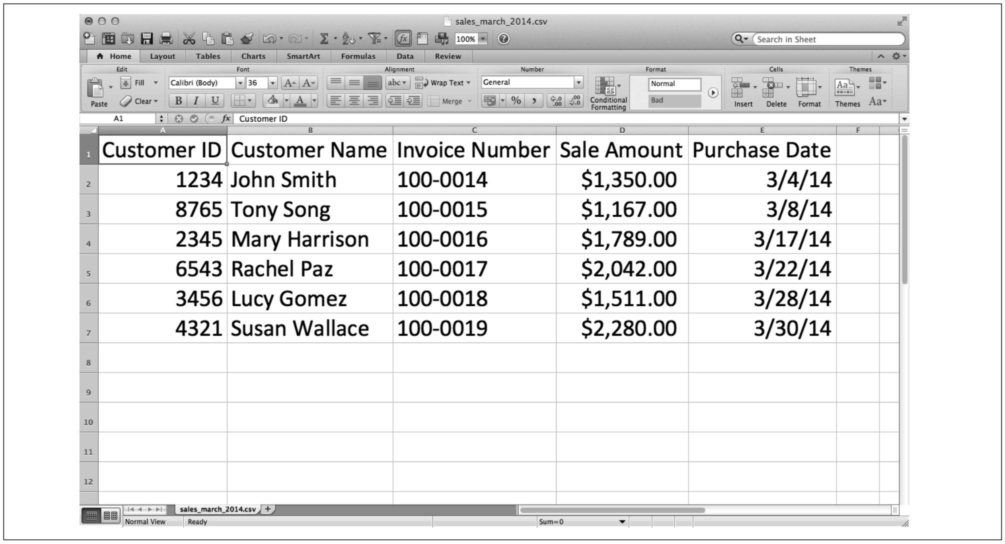
图 2-13：第二个 CSV 文件：sales_february_2014.csv

- 第三个 CSV 文件

  (1) 打开一个电子表格程序。

  (2) 加入图 2-14 所示的数据。

  (3) 将文件保存为 sales_march_2014.csv。



Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
1234	John Smith	100-0014	$1,350.00	3/4/14
8765	Tony Song	100-0015	$1,167.00	3/8/14
2345	Mary Harrison	100-0016	$1,789.00	3/17/14
6543	Rachel Paz	100-0017	$2,042.00	3/22/14
3456	Lucy Gomez	100-0018	$1,511.00	3/28/14
4321	Susan Wallace	100-0019	$2,280.00	3/30/14

图 2-14：第三个 CSV 文件：sales_march_2014.csv

文件计数与文件中的行列计数

下面先从一些简单的行列计数开始。行列计数虽然相当基础，但却是熟悉新数据集的好方式。尽管有些时候你知道要处理的输入文件中的内容，但在多数情况下，文件是别人发送给你的，你不会立即知道是哪些内容。这时，计算一下要处理的文件数量以及每个文件中行与列的数量，会对你有所帮助。

要处理前一节中创建的 3 个 CSV 文件，在文本编辑器中输入下列代码，然后将文件保存为 8csv_reader_counts_for_multiple_files.py：

```
#!/usr/bin/env python3
import csv
import glob
import os
import sys
input_path = sys.argv[1]
file_counter = 0
for input_file in glob.glob(os.path.join(input_path,'sales_*')):
    row_counter = 1
    with open(input_file, 'r', newline='') as csv_in_file:
        filereader = csv.reader(csv_in_file)
        header = next(filereader, None)
        for row in filereader:
            row_counter += 1
```

```
15     print('{0!s}: \t{1:d} rows \t{2:d} columns'.format(\
16 os.path.basename(input_file), row_counter, len(header)))
17      file_counter += 1
18 print('Number of files: {0:d}'.format(file_counter))
```

第 3~4 行代码导入 Python 内置的 glob 和 os 模块，以使我们可以使用它们提供的函数列出和解析你要处理的文件路径名。glob 模块可以定位匹配于某个特定模式的所有路径名。模式中可以包含 Unixshell 风格的通配符，比如 *。在上面这个具体示例中，要搜索的模式是 'sales_*'。这个模式表示要搜索所有文件名以 sales_ 开头并且下划线后面可以是任意字符的文件。因为你创建了 3 个输入文件，所以应该知道使用这段代码可以识别出这 3 个文件，它们的文件名都是以 sales_ 开头的，下划线后面是不同的月份。

以后你可能会想找出一个文件夹下面的所有 CSV 文件，而不是以 sales_ 开头的文件。如果这样，那么你可以简单地将脚本中的模式从 'sales_*' 改变为 '*.csv'。因为 '.csv' 是所有 CSV 文件名末尾的模式，这样做可以有效地找出所有 CSV 文件。

os 模块包含了用于解析路径名的函数。例如，os.path.basename(*path*) 返回 *path* 的基本文件名。即，如果 *path* 是 C:\Users\Clinton\Desktop\my_input_file.csv，那么 os.path.basename(*path*) 返回 my_input_file.csv。

第 8 行代码是将数据处理扩展到多个文件中的关键语句。此行代码创建了一个 for 循环，在一个输入文件集合中迭代，并使用 glob 模块和 os 模块中的函数创建了一个输入文件列表以供处理。这行代码比较复杂，所以需要仔细地分析一下。os 模块中的 os.path.join() 函数将函数圆括号中的两部分连接在一起。input_path 是包含输入文件的文件夹的路径，'sales_*' 代表任何以模式 'sales_' 开头的文件名。

glob 模块中的 glob.glob() 函数将 'sales_*' 中的星号（*）转换为实际的文件名。在这个示例中，glob.glob() 函数和 os.path.join() 函数创建了一个包含 3 个输入文件的列表：

```
['C:\Users\Clinton\Desktop\sales_january_2014.csv',
'C:\Users\Clinton\Desktop\sales_february_2014.csv',
'C:\Users\Clinton\Desktop\sales_march_2014.csv']
```

然后，这行开头的 for 循环语句对于列表中每个输入文件执行下面缩进的各行代码。

第 15 行代码是一个 print 语句，打印出每个输入文件的文件名、文件中的行数、文件中的列数。print 语句中的制表符 \t 不是必需的，但是在各列之间放上一个制表符可以对齐这 3 列。这行代码使用 {} 占位符将 3 个值传入 print 语句。对于第一个值，使用 os.path.basename() 函数从完整路径名中抽取出基本文件名。对于第二个值，使用 row_counter 变量来计算每个输入文件中的总行数。最后，对于第三个值，使用内置的 len 函数计算出列表变量 header 中的值的数量，这个列表变量中包含了每个输入文件的列标题列表。我们使用这个值作为每个输入文件中的列数。最后，在第 15 行代码打印了每个文件的信息之后，第 17 行代码使用 file_counter 变量中的值显示出脚本处理的文件的数量。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 8csv_reader_counts_for_multiple_files.py "C:\Users\Clinton\Desktop"
```

请注意在命令行中，脚本名称后面是一个文件夹路径。在前面的示例中，这个位置都是输

入文件名。在这个示例中，你要处理多个文件，所以必须使用包含所有输入文件的文件夹。

你可以看到 3 个输入文件的文件名和每个文件中的行数与列数被打印到屏幕上。在关于 3 个文件的信息行下面，最后的 `print` 语句显示了之前处理过的输入文件总数。显示信息如图 2-15 所示：

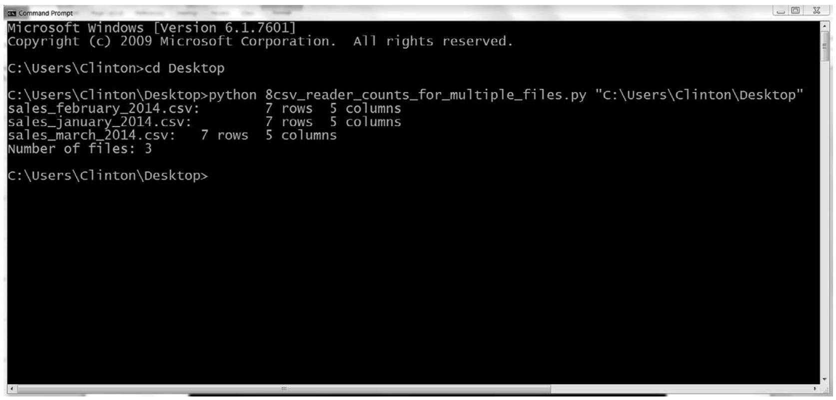


图 2-15：Python 脚本输出：3 个 CSV 文件中的行数与列数

输出结果显示脚本处理了 3 个文件，每个文件都有 7 行和 5 列。

这个示例演示了如何读取多个 CSV 文件并将每个文件的基本信息打印到屏幕上。在你不熟悉要处理的文件时，将要处理文件的基本信息打印出来是非常有用的。知道了输入文件的数量和每个文件中行与列的数量，你就对数据处理的工作量以及文件内容的一致性有了一个大致的概念。

2.7 从多个文件中连接数据

对于包含相似数据的多个文件，你经常希望将其中的数据连接起来，以使所有数据都在一个文件中。以前你完成这种工作的方式可能是：打开每个文件，将每个工作表中的数据复制粘贴到一个单独的工作表中。这种手动处理方式不但浪费时间，还容易出错。而且，在有些情况下，因为需要合并的文件数量和文件大小的原因，手动处理根本不可能完成。

知道了手动连接数据的局限性之后，接下来看一下如何通过 Python 完成这个任务。这里将会使用本节开头创建的 3 个 CSV 文件来演示如何从多个文件中连接数据。

1. 基础Python

要使用基础 Python 将多个输入文件中的数据垂直连接成一个输出文件，在文本编辑器中输入下列代码，然后将文件保存为 9csv_reader_concat_rows_from_multiple_files.py：

```
#!/usr/bin/env python3
import csv
import glob
import os
import sys
```

```
 6 input_path = sys.argv[1]
 7 output_file = sys.argv[2]
 8
 9 first_file = True
10 for input_file in glob.glob(os.path.join(input_path,'sales_*')):
11     print(os.path.basename(input_file))
12     with open(input_file, 'r', newline='') as csv_in_file:
13         with open(output_file, 'a', newline='') as csv_out_file:
14             filereader = csv.reader(csv_in_file)
15             filewriter = csv.writer(csv_out_file)
16             if first_file:
17                 for row in filereader:
18                     filewriter.writerow(row)
19                 first_file = False
20             else:
21                 header = next(filereader, None)
22                 for row in filereader:
23                     filewriter.writerow(row)
```

第 13 行代码是一个 `with` 语句，用来打开输出文件。在前面介绍写入输出文件的示例中，`open` 函数中的字符串是 `'w'`，表示以可写的方式打开输出文件。

在这个示例中，使用 `'a'` 代替 `'w'` 以追加的方式打开输出文件，以使每个输入文件中的数据可以追加（也就是添加）到输出文件中。如果使用可写方式，从一个输入文件中输出的数据会覆盖掉前一个输入文件中的数据，最后的输出文件会只包含最后处理的那个输入文件中的数据。

从第 16 行代码开始的 `if-else` 语句根据第 9 行代码中创建的 `first_file` 变量来区分当前文件是第一个输入文件，还是其后的输入文件。在输入文件中做这个区分的目的是将标题行仅写入输出文件一次。`if` 代码块处理第一个输入文件，将包括标题行的所有行写入输出文件。`else` 代码块处理所有余下的输入文件，使用 `next` 方法将每个文件中的标题行赋给一个变量（这样就可以在后面的处理过程中跳过标题行），然后将其余数据行写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 9csv_reader_concat_rows_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\9output.csv
```

你可以看到输入文件的文件名被打印到屏幕上，如图 2-16 所示。

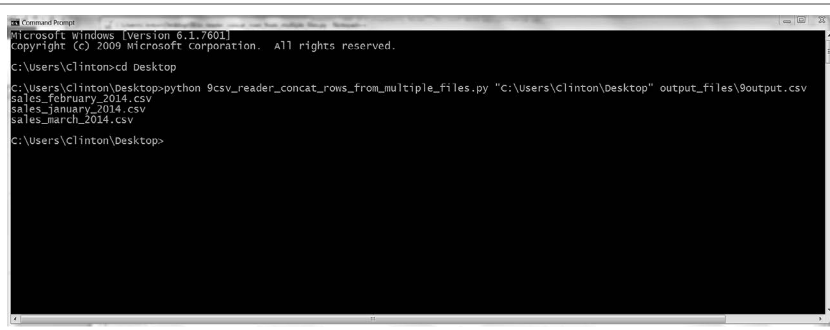
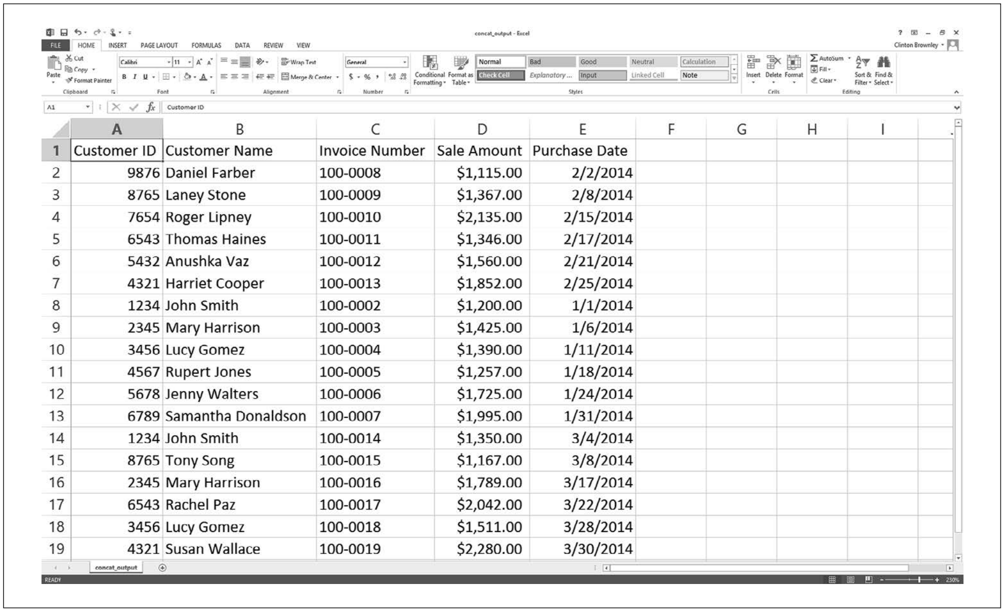


图 2-16：Python 脚本输出：连接成输出文件的文件名

屏幕上的输出展示了处理过的文件的文件名。此外，脚本还将 3 个输入文件中的数据连接成了一个单独的输出文件 9output.csv，位于桌面上的 output_files 文件夹中。图 2-17 显示了文件中的内容。



	A	B	C	D	E
1	Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
2	9876	Daniel Farber	100-0008	$1,115.00	2/2/2014
3	8765	Laney Stone	100-0009	$1,367.00	2/8/2014
4	7654	Roger Lipney	100-0010	$2,135.00	2/15/2014
5	6543	Thomas Haines	100-0011	$1,346.00	2/17/2014
6	5432	Anushka Vaz	100-0012	$1,560.00	2/21/2014
7	4321	Harriet Cooper	100-0013	$1,852.00	2/25/2014
8	1234	John Smith	100-0002	$1,200.00	1/1/2014
9	2345	Mary Harrison	100-0003	$1,425.00	1/6/2014
10	3456	Lucy Gomez	100-0004	$1,390.00	1/11/2014
11	4567	Rupert Jones	100-0005	$1,257.00	1/18/2014
12	5678	Jenny Walters	100-0006	$1,725.00	1/24/2014
13	6789	Samantha Donaldson	100-0007	$1,995.00	1/31/2014
14	1234	John Smith	100-0014	$1,350.00	3/4/2014
15	8765	Tony Song	100-0015	$1,167.00	3/8/2014
16	2345	Mary Harrison	100-0016	$1,789.00	3/17/2014
17	6543	Rachel Paz	100-0017	$2,042.00	3/22/2014
18	3456	Lucy Gomez	100-0018	$1,511.00	3/28/2014
19	4321	Susan Wallace	100-0019	$2,280.00	3/30/2014

图 2-17：输出 CSV 文件，连接了来自于多个输入文件的行

图中显示，脚本成功地连接了来自于 3 个输入文件的数据。输出文件中包含一个标题行和 3 个输入文件中所有的数据行。

在解释代码时，前面提到了在第 13 行代码中为什么要使用 `'a'`（追加模式），而不是 `'w'`（可写模式），还提到了为什么要区分第一个输入文件和其后的输入文件。要实际验证一下的话，你可以将 `'a'` 改成 `'w'`，然后保存脚本，对 3 个输入文件重新执行一下，看看输出文件有什么变化。同样，你也可以删除 `if-else` 语句，将所有输入文件中的所有行都打印出来，看看输出会如何改变。

还有一点需要注意，这个示例中的模式 `'sales_*'` 相对来说是比较特殊的，也就是说在你的桌面上除了 3 个输入文件之外，不太可能还有文件名以 sales_ 开头的文件。其他情况下，你更可能使用一个不那么特殊的模式，比如 `'*.csv'`，来搜索所有 CSV 文件。在这种情况下，你不应该在包含所有输入文件的文件夹内创建输出文件。不应该这样做的原因就是，你在打开输出文件的同时还在处理输入文件。这样，如果你的模式是 `'*.csv'`，输出文件也是个 CSV 文件，那么脚本就会像处理输入文件一样试图处理输出文件，这就会导致问题和错误。这种可能性就是最好在另一个文件夹中处理输出文件的原因，就像在这个示例中所做的一样。

2. pandas

pandas 可以直接从多个文件中连接数据。基本过程就是将每个输入文件读取到 pandas 数据

框中，将所有数据框追加到一个数据框列表，然后使用 concat 函数将所有数据框连接成一个数据框。concat 函数可以使用 axis 参数来设置连接数据框的方式，axis=0 表示从头到尾垂直堆叠，axis=1 表示并排地平行堆叠。

要使用 pandas 将多个输入文件中的数据垂直连接成一个输出文件，在文本编辑器中输入下列代码，然后将文件保存为 pandas_concat_rows_from_multiple_files.py：

```
#!/usr/bin/env python3
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_files = glob.glob(os.path.join(input_path,'sales_*'))
all_data_frames = []
for file in all_files:
    data_frame = pd.read_csv(file, index_col=None)
    all_data_frames.append(data_frame)
data_frame_concat = pd.concat(all_data_frames, axis=0, ignore_index=True)
data_frame_concat.to_csv(output_file, index = False)
```

这段代码垂直堆叠数据框。如果你需要平行连接数据，那么就在 concat 函数中设置 axis=1。除了数据框，pandas 中还有一个数据容器，称为序列。你可以使用同样的语法去连接序列，只是要将连接的对象由数据框改为序列。

有时候，除了简单地垂直或平行连接数据，你还需要基于数据集中的关键字列的值来连接数据集。pandas 提供了类似 SQL join 操作的 merge 函数。如果你很熟悉 SQL join，那么就非常容易理解 merge 函数的语法：pd.merge(DataFrame1, DataFrame2, on='key', how='inner')。

Python 的另一个内置模块 NumPy 也提供了若干函数来垂直或平行连接数据。通常是将 NumPy 导入为 np。然后，要垂直连接数据，你可以使用 np.concatenate([array1, array2], axis=0)、np.vstack((array1, array2)) 或 np.r_[array1, array2]。同样，要平行连接数据，你可以使用 np.concatenate([array1, array2], axis=1)、np.hstack((array1, array2)) 或 np.c_[array1, array2]。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_concat_rows_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

2.8 计算每个文件中值的总和与均值

有些时候，当你有多个输入文件时，需要对每个输入文件计算一些统计量。本节的示例使用之前创建的 3 个 CSV 文件来展示如何计算每个输入文件中某一列的总计和均值。

1. 基础Python

要使用基础 Python 为多个文件计算某列的总计和均值，在文本编辑器中输入下列代码，然

后将文件保存为 10csv_reader_sum_average_from_multiple_files：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import glob
 4 import os
 5 import sys
 6 input_path = sys.argv[1]
 7 output_file = sys.argv[2]
 8 output_header_list = ['file_name', 'total_sales', 'average_sales']
 9 csv_out_file = open(output_file, 'a', newline='')
10 filewriter = csv.writer(csv_out_file)
11 filewriter.writerow(output_header_list)
12 for input_file in glob.glob(os.path.join(input_path,'sales_*')):
13     with open(input_file, 'r', newline='') as csv_in_file:
14          filereader = csv.reader(csv_in_file)
15          output_list = [ ]
16          output_list.append(os.path.basename(input_file))
17          header = next(filereader)
18          total_sales = 0.0
19          number_of_sales = 0.0
20          for row in filereader:
21               sale_amount = row[3]
22               total_sales += float(str(sale_amount).strip('$').replace(',',''))
23               number_of_sales += 1
24          average_sales = '{0:.2f}'.format(total_sales / number_of_sales)
25          output_list.append(total_sales)
26          output_list.append(average_sales)
27          filewriter.writerow(output_list)
28 csv_out_file.close()
```

第 8 行代码创建了一个输出文件的列标题列表。第 10 行代码创建了 filewriter 对象，第 11 行代码将标题行写入输出文件。

第 15 行代码创建了一个空列表，保存要写入输出文件中的每行输出。因为要为每个输入文件计算总计和均值，所以第 16 行代码将输入文件的文件名追加到 output_list 中。

第 17 行代码使用 next 函数除去每个输入文件的标题行。第 18 行代码创建了一个变量 total_sales 并将其初始化为 0。第 20 行代码是一个 for 循环，在每个输入文件的数据行之间迭代。

第 21 行代码使用列表索引取出销售额这列中的值，并赋给变量 sale_amount。第 22 行代码使用 str 函数确保 sale_amount 中的值是一个字符串，然后使用 strip 函数和 replace 函数除去值中的美元符号和逗号。此后使用 float 函数将这个值转换为浮点数，并将这个值加到 total_sales 中的值上。第 23 行代码给 number_of_sales 中的值加 1。

第 24 行代码用 total_sales 中的值除以 number_of_sales 中的值，为输入文件计算出平均销售额，然后将这个数值格式化成具有两位小数的数值，并转换成字符串，赋给变量 average_sales。

第 25 行代码将总销售额作为第二个值添加到 output_list 中。列表中的第一个值是输入文件的名字。这个值在第 16 行代码中被添加到列表中。第 26 行代码将平均销售额作为第三

个值添加到 output_list 中。第 27 行代码将 output_list 中的值写入输出文件。

脚本对每个输入文件都运行这些代码，所以输出文件中会包含对应于每个输入文件的一列文件名、一列总销售额和一列平均销售额。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 10csv_reader_sum_average_from_multiple_files.py \
"C:\Users\Clinton\Desktop" output_files\10output.csv
```

你可以打开输出文件 10output.csv 查看一下结果。

2. pandas

pandas 提供了可以用来计算行和列统计量的摘要统计函数，比如 sum 和 mean。下面的代码演示了如何对于多个文件中的某一列计算这两个统计量（总计和均值），并将每个输入文件的计算结果写入输出文件。

要使用 pandas 计算这两个列统计量，在文本编辑器中输入下列代码，并将文件保存为 pandas_sum_average_from_multiple_files.py：

```
#!/usr/bin/env python3
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_files = glob.glob(os.path.join(input_path,'sales_*'))
all_data_frames = []
for input_file in all_files:
    data_frame = pd.read_csv(input_file, index_col=None)

    total_cost = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
                 for value in data_frame.loc[:, 'Sale Amount']]).sum()

    average_cost = pd.DataFrame([float(str(value).strip('$').replace(',','')) \
                   for value in data_frame.loc[:, 'Sale Amount']]).mean()
    data = {'file_name': os.path.basename(input_file),
            'total_sales': total_sales,
            'average_sales': average_sales}

    all_data_frames.append(pd.DataFrame(data, \
    columns=['file_name', 'total_sales', 'average_sales']))
data_frames_concat = pd.concat(all_data_frames, axis=0, ignore_index=True)
data_frames_concat.to_csv(output_file, index = False)
```

使用列表生成式将销售额这一列中带美元符号的字符串转换为浮点数，然后使用数据框函数将这个对象转换为一个 DataFrame，以便可以使用这两个函数计算列的总计和均值。

因为输出文件中的每行应该包含输入文件名，以及文件中销售额的总计和均值，所以可以将这 3 种数据组合成一个文本框，使用 concat 函数将这些数据框连接成为一个数据框，然后将这个数据框写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_ sum_average_from_multiple_files.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.csv
```

你可以打开输出文件 pandas_output.csv 查看一下结果。

本章介绍了很多基础知识，包括读取和分析 CSV 文件、在 CSV 文件中浏览行与列、处理多个 CSV 文件和为多个 CSV 文件计算统计量的方法。如果你一直跟随本章内容练习示例代码，应该已经完成了 12 个 Python 脚本。

你练习本章中示例代码的最大收获是，它们是浏览和处理文件的基础模块。掌握了本章中的示例代码后，你就可以继续学习如何处理 Excel 文件了，这就是下一章的主题。

2.9 本章练习

(1) 对根据具体条件、集合和正则表达式来筛选行数据的一个脚本进行修改，将与示例代码中不同的一组数据打印出来并写入输出文件。
(2) 对根据索引值或列标题来筛选列数据的一个脚本进行修改，将与示例代码中不同的一组数据打印出来并写入输出文件。
(3) 在一个文件夹中创建一组新的 CSV 输入文件，创建另外一个输出文件夹，使用处理多个文件的一个脚本来处理这些新的输入文件，并将结果写入输出文件夹。

第 3 章

Excel文件

Microsoft Excel 几乎无处不在，使用 Excel 既可以保存客户、库存和雇员数据，还可以跟踪运营、销售和财务活动。人们在商业活动中使用 Excel 的方式五花八门，难以计数。因为 Excel 是商业活动中不可或缺的工具，所以知道如何使用 Python 处理 Excel 数据可以使你将 Python 加入到数据处理工作流中，进而从其他人那里接收数据，并以他们习惯接受的方式分享数据处理结果。

与 Python 的 `csv` 模块不同，Python 中没有处理 Excel 文件（就是带有 .xls 和 .xlsx 扩展名的文件）的标准模块。要完成本章中的示例，你需要 `xlrd` 和 `xlwt` 扩展包。`xlrd` 和 `xlwt` 扩展包使 Python 可以在任何操作系统上处理 Excel 文件，而且对 Excel 日期型数据的支持非常好。如果你安装了 Anaconda Python，那么就已经有了这两个扩展包，因为它们是与安装程序捆绑在一起的。如果你是从 Python.org 网站安装的 Python，那么还需要按照附录 A 中的指示下载并安装这两个扩展包。

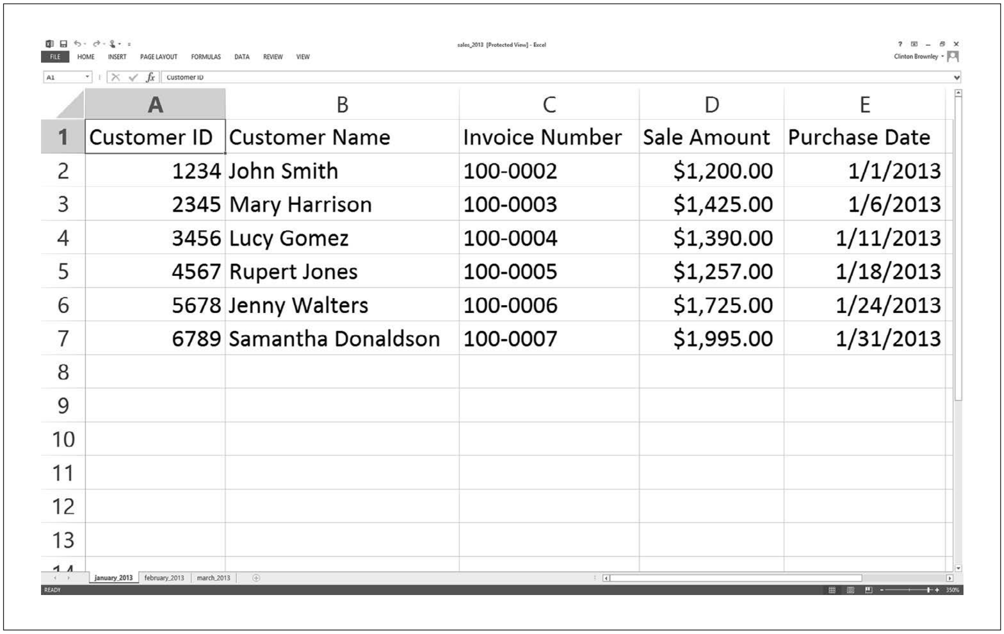
简单解释一下术语：当提到“Excel 文件”时，和“Excel 工作簿”是一回事。Excel 工作簿包含一个或多个 Excel 工作表。在本章中，会交替使用“文件”和“工作簿”这两个词，并将工作簿中的个别工作表直接称为工作表。

和第 2 章中处理 CSV 文件一样，本章先给出使用基础 Python 完成的示例，这样你可以清楚数据处理的每个逻辑步骤，然后给出使用 pandas 完成的示例，这样你可以获得一个（通常情况下）更短小精悍的示例（尽管有一点抽象），你可以对这个示例进行复制和修改，然后用在自己的工作中。

要开始本章示例，需要先创建一个 Excel 工作簿。

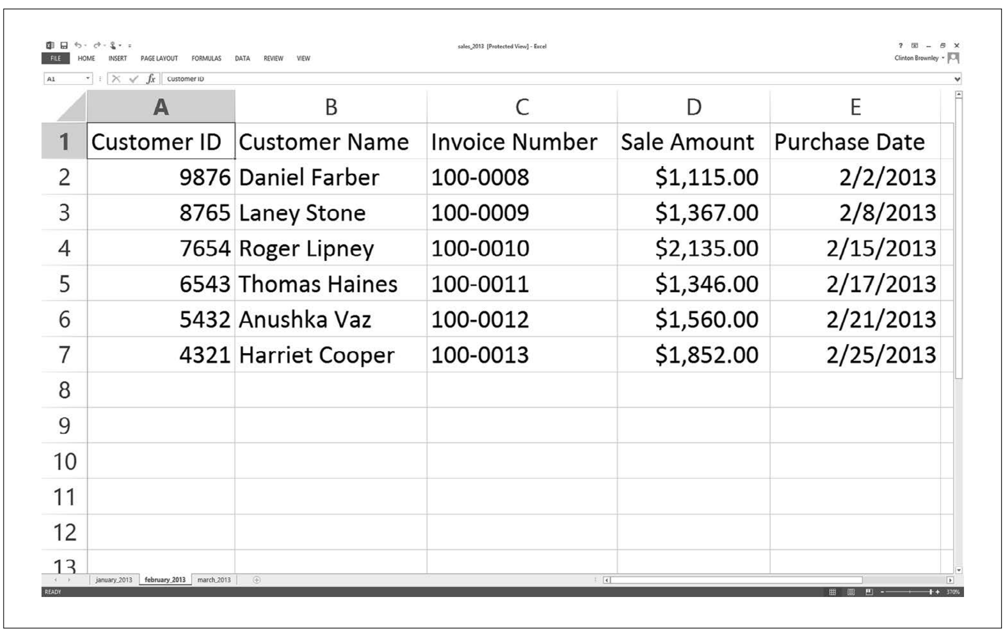
(1) 打开 Microsoft Excel。
(2) 在工作簿中添加 3 个独立的工作表，并分别命名为 january_2013、february_2013 和 march_2013。然后分别添加数据，如图 3-1、图 3-2 和图 3-3 所示。

(3) 将工作簿保存为 sales_2013.xlsx。



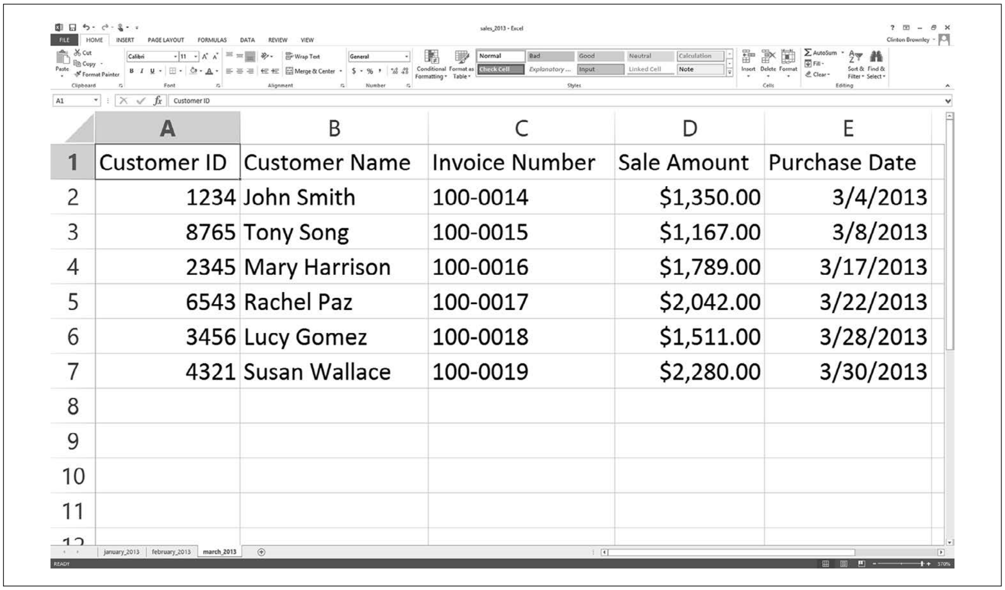
Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
1234	John Smith	100-0002	$1,200.00	1/1/2013
2345	Mary Harrison	100-0003	$1,425.00	1/6/2013
3456	Lucy Gomez	100-0004	$1,390.00	1/11/2013
4567	Rupert Jones	100-0005	$1,257.00	1/18/2013
5678	Jenny Walters	100-0006	$1,725.00	1/24/2013
6789	Samantha Donaldson	100-0007	$1,995.00	1/31/2013

图 3-1：工作表 1：january_2013



Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
9876	Daniel Farber	100-0008	$1,115.00	2/2/2013
8765	Laney Stone	100-0009	$1,367.00	2/8/2013
7654	Roger Lipney	100-0010	$2,135.00	2/15/2013
6543	Thomas Haines	100-0011	$1,346.00	2/17/2013
5432	Anushka Vaz	100-0012	$1,560.00	2/21/2013
4321	Harriet Cooper	100-0013	$1,852.00	2/25/2013

图 3-2：工作表 2：february_2013



Customer ID	Customer Name	Invoice Number	Sale Amount	Purchase Date
1234	John Smith	100-0014	$1,350.00	3/4/2013
8765	Tony Song	100-0015	$1,167.00	3/8/2013
2345	Mary Harrison	100-0016	$1,789.00	3/17/2013
6543	Rachel Paz	100-0017	$2,042.00	3/22/2013
3456	Lucy Gomez	100-0018	$1,511.00	3/28/2013
4321	Susan Wallace	100-0019	$2,280.00	3/30/2013

图 3-3：工作表 3：march_2013

3.1　内省Excel工作簿

既然我们已经有了一个包含 3 个工作表的 Excel 工作簿，那么就开始学习如何在 Python 中处理 Excel 工作簿吧。提示一下，本章中要使用 `xlrd` 和 `xlwt` 扩展包，所以请确认你已经下载并安装了这些扩展包。

你可能已经知道，Excel 文件与 CSV 文件至少在两个重要方面有所不同。首先，与 CSV 文件不同，Excel 文件不是纯文本文件，所以你不能在文本编辑器中打开它并查看数据。为了验证这一点，可以点击刚才创建的 Excel 工作簿并按鼠标右键，然后用一个文本编辑器（比如 Notepad 或者 TextWrangler）打开它。你会看到一堆乱码，而不是正常字符。

其次，与 CSV 文件不同，一个 Excel 工作簿被设计成包含多个工作表，所以你需要知道在不用手动打开工作簿的前提下，如何通过工作簿内省（也就是内部检查）获取其中所有工作表的信息。通过内省一个工作簿，你可以在实际开始处理工作簿中的数据之前，检查工作表的数目和每个工作表中的数据类型和数据量。

内省 Excel 文件有助于确定文件中的数据确实是你需要的，并对数据一致性和完整性做一个初步检查。也就是说，弄清楚输入文件的数量，以及每个文件中的行数和列数，可以使你对数据处理工作的工作量和文件内容的一致性有个大致的概念。

在知道了如何内省工作簿中的工作表之后，下面开始分析单个工作表，然后处理多个工作表和多个工作簿。

要确定工作簿中工作表的数量、名称和每个工作表中行列的数量，在文本编辑器中输入下列代码，然后将文件保存为 1excel_introspect_workbook.py：

```python
#!/usr/bin/env python3
import sys
from xlrd import open_workbook
input_file = sys.argv[1]
workbook = open_workbook(input_file)
print('Number of worksheets:', workbook.nsheets)
for worksheet in workbook.sheets():
    print("Worksheet name:", worksheet.name, "\tRows:",\
        worksheet.nrows, "\tColumns:", worksheet.ncols)
```

图 3-4、图 3-5 和图 3-6 分别展示了 Anaconda Spyder、Notepad++（Windows）和 TextWrangler（macOS）中的脚本。

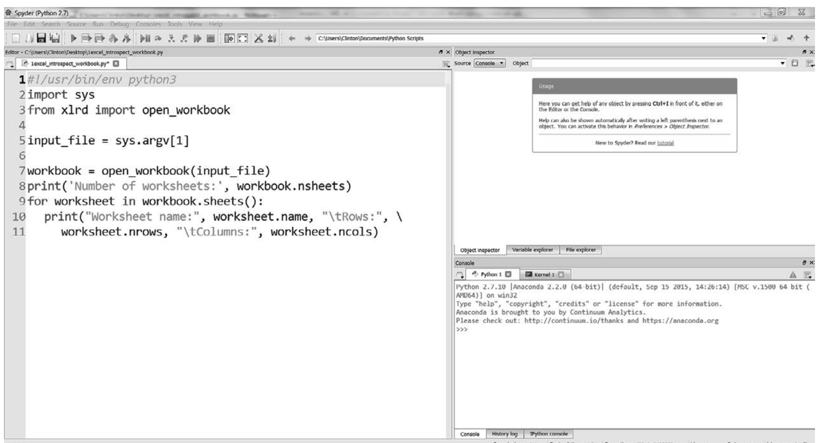


图 3-4：Anaconda Spyder 中的 Python 脚本 1excel_introspect_workbook.py

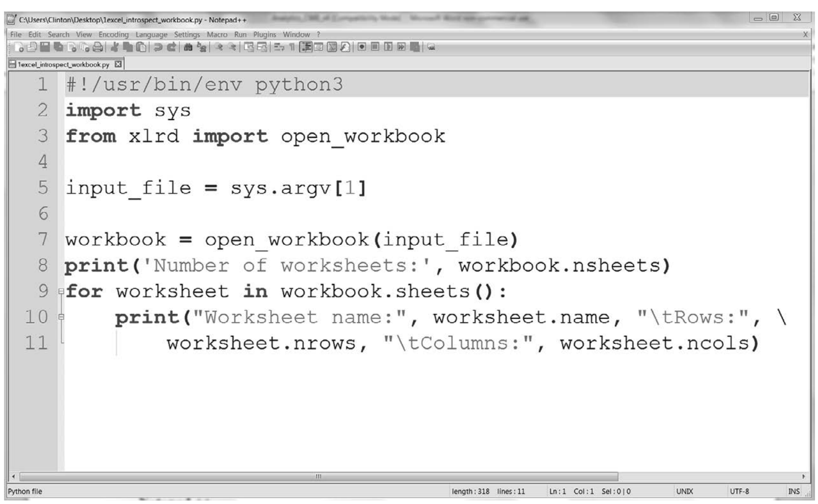


图 3-5：Notepad++（Windows）中的 Python 脚本 1excel_introspect_workbook.py

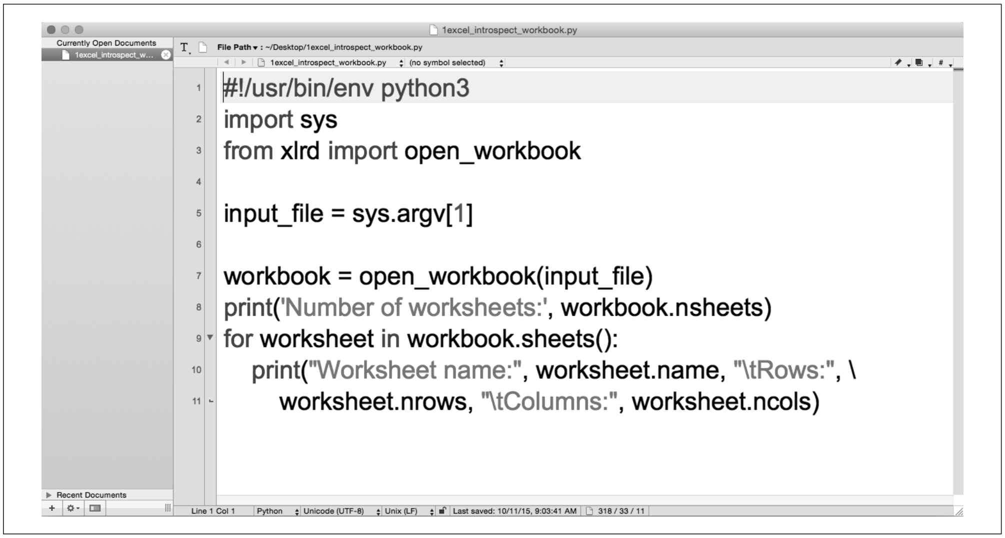


图 3-6：TextWrangler（macOS）中的 Python 脚本 1excel_introspect_workbook.py

第 3 行代码导入 `xlrd` 模块的 `open_workbook` 函数来读取和分析 Excel 文件。

第 7 行代码使用 `open_workbook` 函数打开一个 Excel 输入文件，并赋给一个名为 `workbook` 的对象。`workbook` 对象中包含了工作簿中所有可用的信息，所以可以使用这个对象从工作簿中得到单独的工作表。

第 8 行代码打印出工作簿中工作表的数量。

第 9 行代码是一个 `for` 循环语句，在工作簿中的所有工作表之间迭代。`workbook` 对象的 `sheets` 方法可以识别出工作簿中所有的工作表。

第 10 行代码在屏幕上打印出每个工作表的名称和每个工作表中行与列的数量。`print` 语句使用 `worksheet` 对象的 `name` 属性来确定每个工作表的名称。同样，它使用 `nrows` 和 `ncols` 属性来分别确定每个工作表中行与列的数量。

如果你在 Spyder IDE 中创建了这个文件，按下列步骤运行脚本。

(1) 在 IDE 左上角点击 Run 下拉菜单。
(2) 选择“Configure”。
(3) 当 Run Settings 窗口打开后，选择“Command line options”复选框，然后输入“sales_2013.xlsx”（参见图 3-7）。
(4) 确定“Working directory”是你保存脚本和 Excel 文件的目录。
(5) 点击 Run。

当点击了 Run 按钮（或者是 Run Settings 窗口中的 Run 按钮，或者是 IDE 左上角绿色的 Run 按钮）之后，你会看到输出显示在 IDE 右下角的 Python 控制台窗格上。图 3-7 显示了 Run 下拉菜单、Run Setting 窗口中的关键设置和红框内的输出。

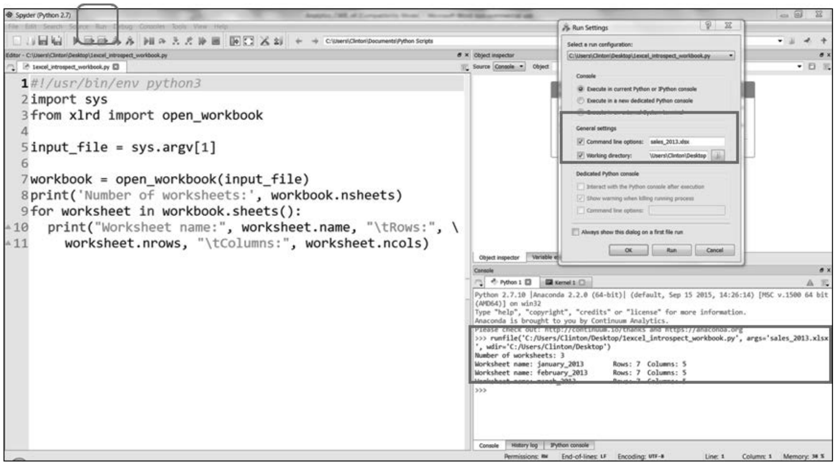


图 3-7：在 Anaconda Spyder 中设置命令行参数

当然，你可以在命令行窗口或终端窗口中运行脚本。要完成这个操作，根据不同的操作系统，使用如下命令。

- Windows 操作系统

```
python 1excel_introspect_workbook.py sales_2013.xlsx
```

- macOS 操作系统

```
chmod +x 1excel_introspect_workbook.py
./1excel_introspect_workbook.py sales_2013.xlsx
```

你可以看到输出被打印到屏幕上，如图 3-8（Windows）或图 3-9（macOS）所示。

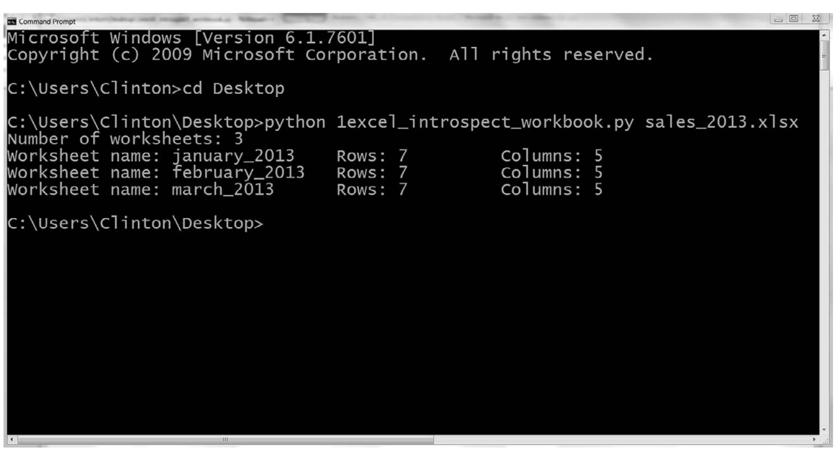


图 3-8：命令行窗口（Windows）中的 Python 脚本输出

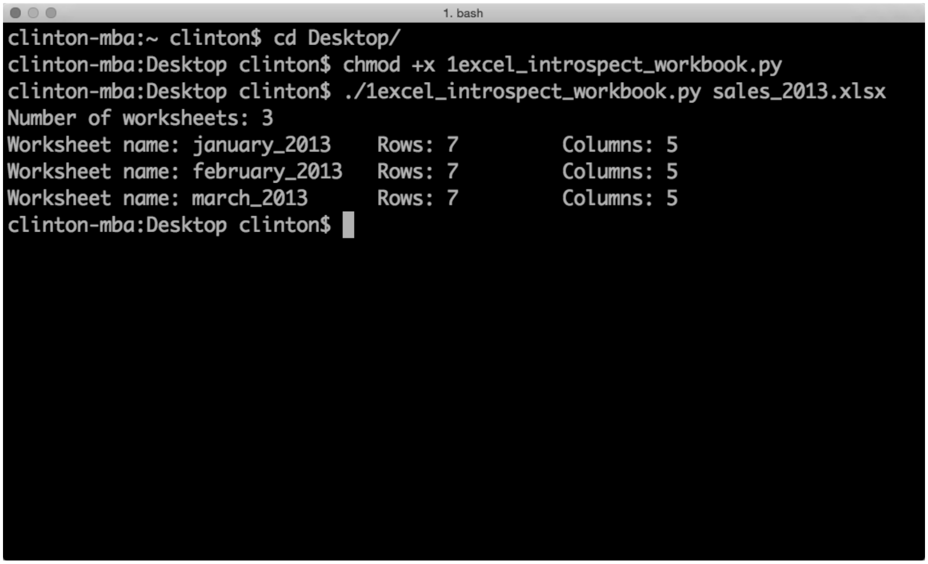


图 3-9：终端窗口（macOS）中的 Python 脚本输出

第一行输出表示 Excel 输入文件 sale__2013.xlsx 中包含 3 个工作表。下面三行说明了 3 个工作表分别名为 january_2013、february_2013 和 march_2013。它们还说明了每个工作表中包含 7 行（包括标题行）和 5 列。

掌握了如何使用 Python 来内省 Excel 工作簿之后，就可以开始学习如何以不同的方法来解析单个工作表了。在此之后，本章会将这种方法扩展为处理多个工作表和多个工作簿。

3.2　处理单个工作表

尽管 Excel 工作簿可以包含多个工作表，有些时候你也只是需要一个工作表中的数据。此外，只要你知道如何分析一个工作表，就可以很容易地扩展到分析多个工作表。

3.2.1　读写Excel文件

基础Python和`xlrd`、`xlwt`模块

要使用基础 Python 和 `xlrd`、`xlwt` 模块读写 Excel 文件，在文本编辑器中输入下列代码，然后将文件保存为 2excel_parsing_and_write.py：

```
#!/usr/bin/env python3
import sys
from xlrd import open_workbook
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')
```

```
 9 with open_workbook(input_file) as workbook:
10     worksheet = workbook.sheet_by_name('january_2013')
11     for row_index in range(worksheet.nrows):
12         for column_index in range(worksheet.ncols):
13             output_worksheet.write(row_index, column_index, \
14                 worksheet.cell_value(row_index, column_index))
15 output_workbook.save(output_file)
```

第 3 行代码导入 xlrd 模块的 open_workbook 函数，第 4 行代码导入 xlwt 模块的 Workbook 对象。

第 7 行代码实例化一个 xlwt Workbook 对象，以使我们可以将结果写入用于输出的 Excel 文件。第 8 行代码使用 xlwt 的 add_sheet 函数为输出工作簿添加一个工作表 jan_2013_output。

第 9 行代码使用 xlrd 的 open_workbook 函数打开用于输入的工作簿，并将结果赋给一个 workbook 对象。第 10 行代码使用这个 workbook 对象的 sheet_by_name 函数引用名称为 january_2013 的工作表。

第 11~12 行代码创建了行与列索引值上的 for 循环语句，使用 range 函数和 worksheet 对象的 nrows 属性和 ncols 属性，在工作表的每行和每列之间迭代。

第 13 行代码使用 xlwt 的 write 函数和行与列的索引将每个单元格的值写入输出文件的工作表。

最后，第 15 行代码保存并关闭输出工作簿。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 2excel_parsing_and_write.py sales_2013.xlsx output_files\2output.xls
```

你可以打开输出文件 2output.xls 查看一下结果。

你可能已经发现，Purchase Date 列（也就是第 E 列）中的日期显示为数值，不是日期。Excel 将日期和时间保存为浮点数，这个浮点数代表从 1900 年 1 月 0 日开始经过的日期数，加上一个 24 小时的小数部分。例如，数值 1 代表 1900 年 1 月 1 日，因为从 1900 年 1 月 0 日过去了 1 天。因此，这一列中的数值代表日期，但是没有格式化为日期的形式。

xlrd 扩展包提供了其他函数来格式化日期值。下一个示例通过演示如何格式化日期数据修正了前一个示例，这样日期值就可以像在输入文件中一样打印到屏幕上或写入输出文件了。

格式化日期数据。这个示例是基于前一个示例的，它展示了如何使用 xlrd 修改日期数据格式，使它们看上去和输入 Excel 文件中一样。例如，如果 Excel 工作表中的一个日期数据为 1/19/2000，那么我们通常希望将 1/19/2000 或其他相关日期格式写入输出文件。但是，就像前一个示例中那样，使用现在的示例代码，你会在输出文件中得到一个数值 36 544.0，因为这就是 1/0/1900 和 1/19/2000 之间的天数。

为了对日期列进行格式化，在文本编辑器中输入下列代码，然后将文件保存为 3excel_parsing_and_write_keep_dates.py：

```
 1 #!/usr/bin/env python3
 2 import sys
 3 from datetime import date
 4 from xlrd import open_workbook, xldate_as_tuple
 5 from xlwt import Workbook
 6 input_file = sys.argv[1]
 7 output_file = sys.argv[2]
 8 output_workbook = Workbook()
 9 output_worksheet = output_workbook.add_sheet('jan_2013_output')
10 with open_workbook(input_file) as workbook:
11     worksheet = workbook.sheet_by_name('january_2013')
12     for row_index in range(worksheet.nrows):
13          row_list_output = []
14          for col_index in range(worksheet.ncols):
15               if worksheet.cell_type(row_index, col_index) == 3:
16                    date_cell = xldate_as_tuple(worksheet.cell_value\
17                         (row_index, col_index),workbook.datemode)
18                    date_cell = date(*date_cell[0:3]).strftime\
19                         ('%m/%d/%Y')
20                    row_list_output.append(date_cell)
21                    output_worksheet.write(row_index, col_index, date_cell)
22               else:
23                    non_date_cell = worksheet.cell_value\
24                         (row_index,col_index)
25                    row_list_output.append(non_date_cell)
26                    output_worksheet.write(row_index, col_index,\
27                         non_date_cell)
28 output_workbook.save(output_file)
```

第 3 行代码从 datetime 模块导入 date 函数，以使我们可以将数值转换成日期并对日期进行格式化。

第 4 行代码从 xlrd 模块中导入两个函数。在前面的示例中，是使用第一个函数打开的 Excel 工作簿，所以这里将重点介绍第二个函数。函数 xldate_as_tuple 可以将 Excel 中代表日期、时间或日期时间的数值转换为元组。只要将数值转换成了元组，就可以提取出具体时间元素（例如：年、月、日）并将时间元素格式化成不同的时间格式（例如：1/1/2010 或 January 1, 2010）。

第 15 行代码创建了一个 if-else 语句来检验单元格类型是否为数字 3。如果你查看了 xlrd 模块的说明文档（https://secure.simplistix.co.uk/svn/xlrd/trunk/xlrd/doc/xlrd.html?p=4966#sheet.Cell-class），就会知道单元格类型为 3 表示这个单元格中包含日期数据。因此，if-else 语句检验每个单元格是否含有日期数据。如果含有日期数据，那么 if 代码块就对单元格进行处理；如果不含有日期数据，那么就使用 else 代码块对单元格进行处理。因为日期数据在最后一列，所以 if 代码块处理最后一列。

第 16 行代码使用 worksheet 对象的 cell_value 函数和行列索引来引用单元格中的值。此外，你还可以使用 cell().value 函数，这两个函数可以给出同样的结果。这个单元格中的值作为 xldate_as_tuple 函数中的第一个参数，会被转换成元组中的一个代表日期的浮点数。

参数 workbook.datemode 是必需的，它可以使函数确定日期是基于 1900 年还是基于 1904 年，并据此将数值转换成正确的元组（在 Mac 上的某些 Excel 版本从 1904 年 1 月 1 日

开始计算日期。要获取这方面的更多信息，请阅读 Microsoft 参考指南（https://support.microsoft.com/en-us/kb/214330）。xldate_as_tuple 函数的结果被赋给一个元组变量 date_cell。这行代码太长了，所以被分为两行，第一行末尾字符是一个反斜杠（你应该记得在第 1 章中我们曾说过反斜杠是必需的，这样 Python 才能将这两行解释为一行）。尽管如此，在你自己的代码中，可以将所有代码都写在一行中而不使用反斜杠。

第 18 行代码使用元组索引来引用元组 date_cell 中的前 3 个元素（也就是年、月、日）并将它们作为参数传给 date 函数，这个函数可以将这些值转换成一个 date 对象，date 对象在第 1 章中曾介绍过。然后，strftime 函数将 date 对象转换为一个具有特定格式的字符串。格式 '%m/%d/%Y' 表示像 2014 年 3 月 15 日这样的日期应该显示为 03/15/2014。格式化后的日期字符串被重新赋给变量 date_cell。第 20 行代码使用列表的 append 函数将 date_cell 中的值追加给输出列表 row_list_output。

在运行了上面的脚本之后，为了对第 16 和 18 行代码中的操作有个大致概念，可以在两个 date_cell=…行之间添加一个 print 语句（也就是 print(date_cell)）。重新保存并运行脚本，看一下 xldate_as_tuple 函数打印在屏幕上的结果。然后，删除这个 print 语句，将它移到第二个 date_cell=…语句下面。重新保存并运行脚本，看一下 date.strftime 函数打印在屏幕上的结果。这些 print 语句可以帮助你看到这两行中的函数是如何将 Excel 中代表日期的数值转换成一个元组，然后又转换成格式化的日期字符串的。

else 代码块处理所有的非日期单元格。第 23 行代码使用 worksheet 对象的 cell_value 函数和行列索引引用单元格中的值，并将其赋给变量 non_date_cell。第 25 行代码使用列表的 append 函数将 non_date_cell 中的值追加给 row_list_output。这两行代码提取出每行前四列中的值，并将它们追加到 row_list_output 中。

在行中的每一列都处理完成，并加入到 row_list_output 中之后，第 26 行代码将 row_list_output 中的值写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 3excel_parsing_and_write_keep_dates.py sales_2013.xlsx\
output_files\3output.xls
```

你可以打开输出文件 3output.xls 查看一下结果。

pandas。pandas 也有一组读取 Excel 文件的函数。下面是使用 pandas 分析 Excel 文件的示例代码。请将这段代码保存为 pandas_read_and_write_excel.py（此段代码读取 Excel 输入文件，将内容打印在屏幕上，然后将内容写入 Excel 输出文件）：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, sheetname='january_2013')
writer = pd.ExcelWriter(output_file)
data_frame.to_excel(writer, sheet_name='jan_13_output', index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_parsing_and_write_keep_dates.py sales_2013.xlsx\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

现在你已经明白了如何处理一个 Excel 工作簿中的工作表，以及如何保留日期格式，那么接下来学习一下如何在工作表中筛选出特定的行。正如我们在第 2 章中做的那样，接下来将讨论如何按照以下方式筛选行：(a) 行中的值满足特定条件；(b) 行中的值属于某个集合；(c) 行中的值匹配于特定的正则表达式。

3.2.2　筛选特定行

有些时候，你并不需要 Excel 文件中的所有行。例如，你可能只需要包含一个特定的词或数值的那些行，或者，你可能只需要那些与一个具体日期相关联的行。在这些情况下，可以使用 Python 筛选掉不需要的行，只保留需要的行。

你可能已经熟悉了如何在 Excel 文件中手动筛选行，但是本章重点在于提高你的能力，使你可以处理因体积太大而难以打开的 Excel 文件，以及手动处理过于浪费时间的多个 Excel 工作表。

1. 行中的值满足某个条件

基础 Python。首先，来看一下如何使用基础 Python 筛选出特定的行。在这个示例中，你想筛选出 Sale Amount 大于 $1400.00 的行。

为了筛选出满足这个条件的行，在文本编辑器中输入下列代码，然后将文件保存为 4excel_value_meets_condition.py：

```
#!/usr/bin/env python3
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')
sale_amount_column_index = 3
with open_workbook(input_file) as workbook:
    worksheet = workbook.sheet_by_name('january_2013')
    data = []
    header = worksheet.row_values(0)
    data.append(header)
    for row_index in range(1,worksheet.nrows):
        row_list = []
        sale_amount = worksheet.cell_value\
        (row_index, sale_amount_column_index)
        if sale_amount > 1400.0:
            for column_index in range(worksheet.ncols):
                cell_value = worksheet.cell_value\
```

```
23                     (row_index,column_index)
24                     cell_type = worksheet.cell_type\
25                     (row_index, column_index)
26                     if cell_type == 3:
27                          date_cell = xldate_as_tuple\
28                          (cell_value,workbook.datemode)
29                          date_cell = date(*date_cell[0:3])\
30                          .strftime('%m/%d/%Y')
31                          row_list.append(date_cell)
32                     else:
33                          row_list.append(cell_value)
34          if row_list:
35               data.append(row_list)
36    for list_index, output_list in enumerate(data):
37         for element_index, element in enumerate(output_list):
38              output_worksheet.write(list_index, element_index, element)
39 output_workbook.save(output_file)
```

第 13 行代码创建了一个空列表 data。我们将用输入文件中要写入输出文件中的那些行来填充这个列表。

第 14 行代码提取出标题行中的值。因为我们想保留标题行，而且检验这一行是否满足筛选条件没有意义，所以第 15 行代码将标题行直接追加到 data 中。

第 18 行代码创建了一个变量 sale_amount，用来保存行中的销售额。cell_value 函数使用第 10 行代码中定义的 sale_amount_column_index 中的值来定位 Sale Amount 列。因为我们想保留销售额大于 $1400.00 的那些行，所以要使用这个变量作为检验条件。

第 21 行代码创建了一个 for 循环，来处理 Sale Amount 大于 1400.0 的那些行。对于这些行，我们先提取出每个单元格的值，赋给变量 cell_value，再提取出每个单元格的类型，赋给变量 cell_type。然后，检验行中的每个值是否是日期类型。如果是日期类型，那么就将这个值格式化成日期数据。为了生成一个每个值都正确格式化的行，我们在第 17 行创建了一个空列表 row_list，然后用第 31 和 33 行代码将行中的日期类型数据和非日期类型数据都追加进 row_list。

我们为输入文件中的每一行都创建空列表 row_list，但是只使用值填充某些空列表（就是 Sale Amount 这列的值大于 1400.0 的那些行的空列表）。所以，对于输入文件中的每一行，第 34 行代码检验 row_list 是否为空，只将非空的 row_list 添加到 data 中。

最后，在第 36 和 37 行代码中，我们在 data 中的各个列表之间和列表中的各个值之间进行迭代，将这些值写入输出文件。将要保留的行追加到一个新列表 data 中的原因是，这样可以得到新的连续的行索引值。于是，当我们将这些行写入输出文件时，它们看上去就像是一个连续的整体，行与行之间不会出现缺口。否则，如果在主体 for 循环中处理各行的时候就将它们写入输出文件的话，那么 xlwt 的 write 函数就会使用输入文件中原来的行索引值将行写入输出文件，造成行与行之间存在缺口。在后面选择特定列的小节中，我们还会使用这种方法，以此来保证将各列作为一个连续整体写入输出文件，列与列之间不出现缺口。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 4excel_value_meets_condition.py sales_2013.xlsx output_files\4output.xls
```

你可以打开输出文件 4output.xls 查看一下结果。

pandas。你可以使用 pandas 筛选出符合某个条件的行，指定你想判断的列的名称，并在数据框名称后面的方括号中设定具体的判断条件。例如，在下面的脚本中，我们设定的判断条件就可以筛选出 Sale Amount 列大于 1400.00 的所有行。

如果你需要设定多个条件，那么可以将这些条件放在圆括号中，根据需要的逻辑顺序用“&”或“|”连接起来。在注释掉的两行代码中，展示了如何基于两个条件来筛选行。第一行代码使用“&”，表示两个条件必须都为真。第二行代码使用“|”，表示只要一个条件为真就可以。（在下面的示例代码中，并没有加了注释的语句，应该是作者删掉了。——译者注）

要使用 pandas 筛选出满足特定条件的行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_meets_condition.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)
data_frame_value_meets_condition = \
    data_frame[data_frame['Sale Amount'].astype(float) > 1400.0]
writer = pd.ExcelWriter(output_file)
data_frame_value_meets_condition.to_excel(writer, sheet_name='jan_13_output',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_value_meets_condition.py sales_2013.xlsx\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

2. 行中的值属于某个集合

基础 Python。要使用基础 Python 筛选出购买日期属于一个特定集合（例如：日期 01/24/2013 和 01/31/2013 的集合）的行，在文本编辑器中输入下列代码，并将文件保存为 5excel_value_in_set.py：

```
#!/usr/bin/env python3
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')
```

```
10 important_dates = ['01/24/2013', '01/31/2013']
11 purchase_date_column_index = 4
12 with open_workbook(input_file) as workbook:
13     worksheet = workbook.sheet_by_name('january_2013')
14     data = []
15     header = worksheet.row_values(0)
16     data.append(header)
17     for row_index in range(1, worksheet.nrows):
18          purchase_datetime = xldate_as_tuple(worksheet.cell_value\
19          (row_index, purchase_date_column_index)\
20          ,workbook.datemode)
21          purchase_date = date(*purchase_datetime[0:3]).strftime('%m/%d/%Y')
22          row_list = []
23          if purchase_date in important_dates:
24          for column_index in range(worksheet.ncols):
25               cell_value = worksheet.cell_value\
26               (row_index,column_index)
27               cell_type = worksheet.cell_type(row_index, column_index)
28               if cell_type == 3:
29                    date_cell = xldate_as_tuple\
30                    (cell_value,workbook.datemode)
31                   date_cell = date(*date_cell[0:3])\
32                   .strftime('%m/%d/%Y')
33                   row_list.append(date_cell)
34               else:
35                   row_list.append(cell_value)
36      if row_list:
37           data.append(row_list)
38    for list_index, output_list in enumerate(data):
39         for element_index, element in enumerate(output_list):
40              output_worksheet.write(list_index, element_index, element)
41 output_workbook.save(output_file)
```

这个脚本与基于条件筛选行的脚本非常相似，区别在于第 10、21 和 23 行。第 10 行代码创建了一个列表 `important_dates`，包含了要使用的日期。第 21 行代码创建了一个变量 `purchase_date`，它等于 Purchase Date 列中格式化后的值，并用它来匹配 `important_dates` 中格式化的日期。第 23 行代码检验行中的日期是否是 `important_dates` 中的一个日期。如果是，那么就处理这一行，并将其写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 5excel_value_in_set.py sales_2013.xlsx output_files\5output.xls
```

你可以打开输出文件 5output.xls 查看一下结果。

pandas。在这个示例中，我们想筛选出购买日期为 01/24/2013 或 01/31/2013 的行。pandas 提供了 `isin` 函数，你可以使用它来检验一个特定值是否在一个列表中。

要使用 pandas 基于集合成员筛选行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_in_set.py：

```
#!/usr/bin/env python3
```

```
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)
important_dates = ['01/24/2013','01/31/2013']
data_frame_value_in_set = data_frame[data_frame['PurchaseDate']\
.isin(important_dates)]
writer = pd.ExcelWriter(output_file)
data_frame_value_in_set.to_excel(writer, sheet_name='jan_13_output', index=False)
writer.save()
```

在命令行中运行这个脚本：

```
python pandas_value_in_set.py sales_2013.xlsx output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3. 行中的值匹配于特定模式

基础 Python。要使用基础 Python 筛选出客户姓名包含一个特定模式（例如：以大写字母 J 开头）的行，在文本编辑器中输入下列代码，然后将文件保存为 6excel_value_matches_pattern.py：

```
#!/usr/bin/env python3
import re
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')
pattern = re.compile(r'(?P<my_pattern>^J.*)')
customer_name_column_index = 1
with open_workbook(input_file) as workbook:
    worksheet = workbook.sheet_by_name('january_2013')
    data = []
    header = worksheet.row_values(0)
    data.append(header)
    for row_index in range(1, worksheet.nrows):
         row_list = []
         if pattern.search(worksheet.cell_value\
         (row_index, customer_name_column_index)):
              for column_index in range(worksheet.ncols):
                   cell_value = worksheet.cell_value\
                   (row_index,column_index)
                   cell_type = worksheet.cell_type(row_index, column_index)
                   if cell_type == 3:
                        date_cell = xldate_as_tuple\
                        (cell_value,workbook.datemode)
                        date_cell = date(*date_cell[0:3])\
                        .strftime('%m/%d/%Y')
                        row_list.append(date_cell)
```

```
32                  else:
33                      row_list.append(cell_value)
34          if row_list:
35              data.append(row_list)
36  for list_index, output_list in enumerate(data):
37      for element_index, element in enumerate(output_list):
38          output_worksheet.write(list_index, element_index, element)
39 output_workbook.save(output_file)
```

第 2 行代码导入 re 模块，以使我们可以使用模块中的函数和方法。

第 11 行代码使用 re 模块的 compile 函数创建了一个正则表达式 pattern。如果你能看懂，那么这个函数中的内容就很好解释。r 表示单引号之间的模式是一个原始字符串。元字符 ?P<my_pattern> 捕获了名为 <my_pattern> 的组中匹配了的子字符串，以便在需要时将它们打印到屏幕上或写入文件。我们要搜索的实际模式是 '^J.*'。插入符号（^）是一个特殊符号，表示“在字符串开头搜索模式”。所以，字符串需要以大写字母 J 开头。句点 . 可以匹配任何字符，除了换行符。所以除换行符之外的任何字符都可以跟在 J 后面。最后，* 表示重复前面的字符 0 次或更多次。.* 组合在一起用来表示除换行符之外的任意字符可以在 J 后面出现任意次。

第 20 行代码使用 re 模块中的 search 函数在 Customer Name 列中搜索模式，并检测是否能找到一个匹配。如果找到了一个匹配，就将这一行中的每个值添加到 row_list 中。第 31 行代码将日期值添加到 row_list 中，第 33 行代码将非日期值添加到 row_list 中。如果 row_list 不是空的，第 35 行代码将 row_list 中的每个列表值添加到 data。

最后，第 36 和 37 行代码中的两个 for 循环在 data 中的各个列表中迭代，将各行写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 6excel_value_matches_pattern.py sales_2013.xlsx output_files\6output.xls
```

你可以打开输出文件 6output.xls 查看一下结果。

pandas。在这个示例中，你想筛选出客户姓名以大写字母 J 开头的那些行。pandas 提供了若干字符串和正则表达式函数，包括 startswith、endswith、match 和 search 等，你可以使用这些函数在文本中识别子字符串和模式。

要使用 pandas 筛选出客户姓名以大写字母 J 开头的那些行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_matches_pattern.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)
data_frame_value_matches_pattern = data_frame[data_frame['Customer Name']\
.str.startswith("J")]
writer = pd.ExcelWriter(output_file)
```

```
data_frame_value_matches_pattern.to_excel(writer, sheet_name='jan_13_output',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_value_matches_pattern.py sales_2013.xlsx\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.2.3　选取特定列

有些时候，你并不需要工作表中所有的列。在这种情况下，可以使用 Python 选取出你需要保留的列。

有两种通用方法可以在 Excel 文件中选取特定的列。下面的小节演示了这两种选取列的方法：

- 使用列索引值
- 使用列标题

1. 列索引值

基础 Python。从工作表中选取特定列的一种方法是使用要保留的列的索引值。当你想保留的列的索引值非常容易识别，或者在处理多个输入文件过程中，各个输入文件中列的位置是一致（也就是不会发生改变）的时候，这种方法非常有效。

例如，假设我们想保留 Customer Name 和 Purchase Date 这两列。要使用基础 Python 选取这两列，在文本编辑器中输入下列代码，然后将文件保存为 7excel_column_by_index.py：

```
#!/usr/bin/env python3
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('jan_2013_output')
my_columns = [1, 4]
with open_workbook(input_file) as workbook:
    worksheet = workbook.sheet_by_name('january_2013')
    data = []
    for row_index in range(worksheet.nrows):
        row_list = []
        for column_index in my_columns:
            cell_value = worksheet.cell_value(row_index,column_index)
            cell_type = worksheet.cell_type(row_index, column_index)
            if cell_type == 3:
                date_cell = xldate_as_tuple\
                (cell_value,workbook.datemode)
                date_cell = date(*date_cell[0:3]).strftime('%m/%d/%Y')
```

```
23                    row_list.append(date_cell)
24                else:
25                    row_list.append(cell_value)
26           data.append(row_list)
27    for list_index, output_list in enumerate(data):
28         for element_index, element in enumerate(output_list):
29              output_worksheet.write(list_index, element_index, element)
30 output_workbook.save(output_file)
```

第 10 行代码创建了一个列表变量 `my_columns`，包含整数 1 和 4。这两个整数分别代表 Customer Name 和 Purchase Date 列的索引值。

第 16 行代码创建了一个 `for` 循环，在 `my_columns` 中的两个列索引值之间迭代。在每次循环中，提取出列中单元格的值和类型，判断单元格中的值是否是日期类型，并对单元格进行相应处理，然后将值追加到 `row_list` 中。第 26 行代码将 `row_list` 中的值添加到 `data` 中。

最后，第 27 和 28 行代码中的两个 `for` 循环在 `data` 中的列表之间迭代，将其中的值写入输出文件。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 7column_column_by_index.py sales_2013.xlsx output_files\7output.xls
```

你可以打开输出文件 7output.xls 查看一下结果。

pandas。有很多方法可以使用 pandas 选取特定列。一种方法是设置数据框，在方括号中列出要保留的列的索引值或名称（字符串）。

另一种方法，也就是下面所展示的，是设置数据框和 `iloc` 函数。`iloc` 函数非常有用，因为它可以使你同时选择特定的行与特定的列。所以，如果使用 `iloc` 函数来选择列，那么就需要在列索引值前面加上一个冒号和一个逗号，表示你想为这些特定的列保留所有的行。否则，`iloc` 函数也会使用这些索引值去筛选行。

要使用 pandas 基于索引值去选取列，在文本编辑器中输入下列代码，然后将文件保存为 pandas_column_by_index.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)
data_frame_column_by_index = data_frame.iloc[:, [1, 4]]
writer = pd.ExcelWriter(output_file)
data_frame_column_by_index.to_excel(writer, sheet_name='jan_13_output',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_column_by_index.py sales_2013.xlsx output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

2. 列标题

第二种在工作表中选取一组列的方法是使用列标题。当你想保留的列的标题非常容易识别，或者在处理多个输入文件过程中，各个输入文件中列的位置会发生改变，但标题不变的时候，这种方法非常有效。

基础 Python。要使用基础 Python 选取 Customer ID 和 Purchase Date 列，在文本编辑器中输入下列代码，然后将文件保存为 8excel_column_by_name.py：

```
 1 #!/usr/bin/env python3
 2 import sys
 3 from datetime import date
 4 from xlrd import open_workbook, xldate_as_tuple
 5 from xlwt import Workbook
 6 input_file = sys.argv[1]
 7 output_file = sys.argv[2]
 8 output_workbook = Workbook()
 9 output_worksheet = output_workbook.add_sheet('jan_2013_output')
10 my_columns = ['Customer ID', 'Purchase Date']
11 with open_workbook(input_file) as workbook:
12     worksheet = workbook.sheet_by_name('january_2013')
13     data = [my_columns]
14     header_list = worksheet.row_values(0)
15     header_index_list = []
16     for header_index in range(len(header_list)):
17          if header_list[header_index] in my_columns:
18               header_index_list.append(header_index)
19     for row_index in range(1,worksheet.nrows):
20         row_list = []
21         for column_index in header_index_list:
22              cell_value = worksheet.cell_value(row_index,column_index)
23              cell_type = worksheet.cell_type(row_index, column_index)
24              if cell_type == 3:
25                   date_cell = xldate_as_tuple\
26                   (cell_value,workbook.datemode)
27                   date_cell = date(*date_cell[0:3]).strftime('%m/%d/%Y')
28                   row_list.append(date_cell)
29              else:
30                   row_list.append(cell_value)
31         data.append(row_list)
32    for list_index, output_list in enumerate(data):
33         for element_index, element in enumerate(output_list):
34               output_worksheet.write(list_index, element_index, element)
35 output_workbook.save(output_file)
```

第 10 行代码创建了一个列表变量 my_columns，包含要保留的两列的名称。因为这是要写入输出文件的列标题，所以在第 13 行代码中，直接将其加入输出列表 data。

第 16 行代码创建了一个 for 循环，在 header_list 中的列标题索引值之间迭代。第 17 行代码使用列表索引来检验每个列标题是否在列表 my_columns 中。如果是，就使用第 18 行代码将这个列标题的索引值追加到 header_index_list 中。后面将在第 21 行代码中使用这

些索引值，仅处理那些要写入输出文件的列。

第 21 行代码创建了一个 for 循环，在 header_index_list 中的列索引值之间迭代。通过使用 header_index_list，只处理在 my_columns 中列出的那些列。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 8excel_column_by_name.py sales_2013.xlsx output_files\8output.xls
```

你可以打开输出文件 8output.xls 查看一下结果。

pandas。要使用 pandas 基于列标题选取特定列，一种方式是在数据框名称后面的方括号中将列名以字符串方式列出。另外一种方式是使用 loc 函数。如果使用 loc 函数，那么需要在列标题列表前面加上一个冒号和一个逗号，表示你想为这些特定的列保留所有行。

要使用 pandas 基于列标题选取列，在文本编辑器中输入下列代码，然后将文件保存为 pandas_column_by_name.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, 'january_2013', index_col=None)
data_frame_column_by_name = data_frame.loc[:, ['Customer ID', 'Purchase Date']]
writer = pd.ExcelWriter(output_file)
data_frame_column_by_name.to_excel(writer, sheet_name='jan_13_output',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_column_by_name.py sales_2013.xlsx output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.3 读取工作簿中的所有工作表

本章到目前为止，都在演示如何处理单个工作表。有些时候，你只需要处理一个工作表就可以了。在这些情况下，这里的示例可以告诉你如何使用 Python 程序去自动处理工作表。

但是，在很多情况下你需要处理多个工作表，多到使用手工处理效率非常低或者根本不可行。在这种情况下，Python 会给你惊喜，因为它可以让你自动化和规模化地进行数据处理，远远超过手工处理能够达到的限度。本小节提供了两个示例，演示了如何在一个工作簿的所有工作表中筛选特定的行与列。

我仅提供一个用于筛选行的示例和一个用于筛选列的示例，因为我想把本章篇幅保留在合理范围之内（后面还会有介绍如何处理一个工作簿中特定的一组工作表和如何处理多个工作簿的章节）。此外，在前面的示例中，你已经掌握了选择特定的行与列的其他方法，就

应该知道如何将那些筛选操作应用在这里的示例中。

3.3.1 在所有工作表中筛选特定行

1. 基础Python

要使用基础 Python 在所有工作表中筛选出销售额大于 $2000.00 的所有行，在文本编辑器中输入下列代码，然后将文件保存为 9excel_value_meets_condition_all_worksheets.py：

```
 1 #!/usr/bin/env python3
 2 import sys
 3 from datetime import date
 4 from xlrd import open_workbook, xldate_as_tuple
 5 from xlwt import Workbook
 6 input_file = sys.argv[1]
 7 output_file = sys.argv[2]
 8 output_workbook = Workbook()
 9 output_worksheet = output_workbook.add_sheet('filtered_rows_all_worksheets')
10 sales_column_index = 3
11 threshold = 2000.0
12 first_worksheet = True
13 with open_workbook(input_file) as workbook:
14     data = []
15     for worksheet in workbook.sheets():
16          if first_worksheet:
17               header_row = worksheet.row_values(0)
18               data.append(header_row)
19               first_worksheet = False
20          for row_index in range(1,worksheet.nrows):
21               row_list = []
22               sale_amount = worksheet.cell_value\
23               (row_index, sales_column_index)
24               if sale_amount > threshold:
25                    for column_index in range(worksheet.ncols):
26                         cell_value = worksheet.cell_value\
27                         (row_index,column_index)
28                         cell_type = worksheet.cell_type\
29                         (row_index, column_index)
30                         if cell_type == 3:
31                              date_cell = xldate_as_tuple\
32                              (cell_value,workbook.datemode)
33                              date_cell = date(*date_cell[0:3])\
34                              .strftime('%m/%d/%Y')
35                              row_list.append(date_cell)
36                         else:
37                              row_list.append(cell_value)
38               if row_list:
39                    data.append(row_list)
40    for list_index, output_list in enumerate(data):
41         for element_index, element in enumerate(output_list):
42               output_worksheet.write(list_index, element_index, element)
43 output_workbook.save(output_file)
```

第 10 行代码创建了一个变量 `sales_column_index`，保存 Sale Amount 列的索引值。同样，

第 11 行代码创建了一个变量 threshold 来保存你所关心的销售额。我们要将 Sale Amount 列中的每个值与这个阈值进行比较，来确定哪一行要被写入到输出文件中。

第 15 行代码创建了一个 for 循环，用来在工作簿中的所有工作表之间迭代。它使用 workbook 对象的 sheets 属性来列出工作簿中所有的工作表。

第 16 行代码判断当前工作表是不是第一个工作表，如果是第一个工作表，我们就提取出标题行，将其追加到 data 中，然后将 first_worksheet 设为 False。代码继续处理余下的销售额大于阈值的数据行。

对于所有后续的工作表，first_worksheet 都是 False，所以脚本直接来到第 20 行代码处理每个工作表中的数据行。因为 range 函数不是从 0 开始，而是从 1 开始，所以你应该知道代码处理的是数据行，不是标题行。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 9excel_value_meets_condition_all_worksheets.py sales_2013.xlsx\
output_files\9output.xls
```

你可以打开输出文件 9output.xls 查看一下结果。

2. pandas

在 pandas 中，通过在 read_excel 函数中设置 sheetname=None，可以一次性读取工作簿中的所有工作表。pandas 将这些工作表读入一个数据框字典，字典中的键就是工作表的名称，值就是包含工作表中数据的数据框。所以，通过在字典的键和值之间迭代，你可以使用工作簿中所有的数据。当你在每个数据框中筛选特定行时，结果是一个新的筛选过的数据框，所以你可以创建一个列表保存这些筛选过的数据框，然后将它们连接成一个最终数据框。

在下面这个示例中，我们想在所有工作表中筛选出销售额大于 $2000.00 的所有行。要使用 pandas 筛选出这些行，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_meets_condition_all_worksheets.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, sheetname=None, index_col=None)
row_output = []
for worksheet_name, data in data_frame.items():
    row_output.append(data[data['Sale Amount'].astype(float) > 2000.0])
filtered_rows = pd.concat(row_output, axis=0, ignore_index=True)
writer = pd.ExcelWriter(output_file)
filtered_rows.to_excel(writer, sheet_name='sale_amount_gt2000', index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_value_meets_condition_all_worksheets.py sales_2013.xlsx\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.3.2 在所有工作表中选取特定列

有些时候，Excel 工作簿中包含了多个工作表，每个工作表中包含的列并不都是你需要的。在这种情况下，你可以使用 Python 读取所有工作表，筛选掉不需要的列，只保留需要的列。

从前面的内容可知，至少有两种方法可以从工作表中选取一组列：使用列索引值和列标题。下面的示例演示了如何使用列标题从一个工作簿的所有工作表中选取特定的列。

1. 基础Python

要使用基础 Python 在所有工作表中选取 Customer Name 和 Sale Amount 列，在文本编辑器中输入下列代码，然后将文件保存为 10excel_column_by_name_all_worksheet.py：

```
1 #!/usr/bin/env python3
2 import sys
3 from datetime import date
4 from xlrd import open_workbook, xldate_as_tuple
5 from xlwt import Workbook
6 input_file = sys.argv[1]
7 output_file = sys.argv[2]
8 output_workbook = Workbook()
9 output_worksheet = output_workbook.add_sheet('selected_columns_all_worksheets')
10 my_columns = ['Customer Name', 'Sale Amount']
11 first_worksheet = True
12 with open_workbook(input_file) as workbook:
13    data = [my_columns]
14    index_of_cols_to_keep = []
15    for worksheet in workbook.sheets():
16         if first_worksheet:
17              header = worksheet.row_values(0)
18              for column_index in range(len(header)):
19                   if header[column_index] in my_columns:
20                        index_of_cols_to_keep.append(column_index)
21              first_worksheet = False
22        for row_index in range(1, worksheet.nrows):
23             row_list = []
24             for column_index in index_of_cols_to_keep:
25                  cell_value = worksheet.cell_value\
26                  (row_index, column_index)
27                  cell_type = worksheet.cell_type(row_index, column_index)
28                  if cell_type == 3:
29                       date_cell = xldate_as_tuple\
30                       (cell_value,workbook.datemode)
31                       date_cell = date(*date_cell[0:3])\
32                       .strftime('%m/%d/%Y')
33                       row_list.append(date_cell)
34                  else:
35                       row_list.append(cell_value)
36              data.append(row_list)
37    for list_index, output_list in enumerate(data):
38         for element_index, element in enumerate(output_list):
39               output_worksheet.write(list_index, element_index, element)
40 output_workbook.save(output_file)
```

第 10 行代码创建了一个列表变量 `my_columns`，包含了我们要保留的两列的名称。

第 13 行代码将 my_columns 放入 data，作为 data 中的第一个列表，因为它是要写入输出文件的列的列标题。第 14 行代码创建了一个空列表 index_of_cols_to_keep，用来保存 Customer Name 和 Sale Amount 列的索引值。

第 16 行代码检验当前是否在处理第一个工作表。如果是第一个工作表，我们就识别出 Customer Name 和 Sale Amount 列的索引值，并将其追加到列表 index_of_cols_to_keep 中。然后，将 first_worksheet 的值设为 False。代码继续处理余下的数据行，第 24 行代码仅用于处理 Customer Name 和 Sale Amount 列中的值。

对于所有后续的工作表，first_worksheet 都是 False，所以脚本直接来到第 22 行代码处理每个工作表中的数据行。对于这些工作表，只处理索引值在 index_of_cols_to_keep 中的那些列。如果这些列中有日期型数据，就将其格式化。在组合好一行要写入输出文件的数据之后，使用第 36 行代码将这个数据列表追加到 data 中。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 10excel_column_by_name_all_worksheets.py sales_2013.xlsx\
output_files\10output.xls
```

你可以打开输出文件 10output.xls 查看一下结果。

2. pandas

我们再一次使用 pandas 中的 read_excel 函数将所有工作表读入一个字典。然后，使用 loc 函数在每个工作表中选取特定的列，创建一个筛选过的数据框列表，并将这些数据框连接在一起，形成一个最终数据框。

在这个示例中，我们想在所有工作表中选取 Customer Name 和 Sale Amount 列。要使用 pandas 选取这些列，在文本编辑器中输入下列代码，然后将文件保存为 pandas_column_by_name_all_worksheets.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_excel(input_file, sheetname=None, index_col=None)
column_output = []
for worksheet_name, data in data_frame.items():
    column_output.append(data.loc[:, ['Customer Name', 'Sale Amount']])
selected_columns = pd.concat(column_output, axis=0, ignore_index=True)
writer = pd.ExcelWriter(output_file)
selected_columns.to_excel(writer, sheet_name='selected_columns_all_worksheets',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_column_by_name_all_worksheets.py sales_2013.xlsx\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.4 在Excel工作簿中读取一组工作表

本章最开始的几小节演示了如何在一个工作表中筛选特定的行与特定的列。前一个小节演示了如何在一个工作簿的所有工作表中筛选特定的行与列。

但是，有些情况下，你只需要处理工作簿中的一组工作表。例如，你的工作簿可能包含很多工作表，但是你只需要处理其中的 20 个。在这种情况下，可以使用工组簿的 `sheet_by_index` 或 `sheet_by_name` 函数来处理一组工作表。

这一小节只提供了一个示例来演示如何在工作簿的一组工作表中筛选特定的行。之所以这样做，是因为到目前为止，你应该能够将前面各个示例中的筛选与选择操作集成到这个示例中了。

在一组工作表中筛选特定行

1. 基础Python

在这个示例中，我们想从第一个和第二个工作表中筛选出销售额大于 $1900.00 的那些行。要使用基础 Python 从第一个和第二个工作表中筛选出这样的行，在文本编辑器中输入下列代码，然后将文件保存为 11excel_value_meets_condition_set_of_worksheets.py：

```
#!/usr/bin/env python3
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_file = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('set_of_worksheets')
my_sheets = [0,1]
threshold = 1900.0
sales_column_index = 3
first_worksheet = True
with open_workbook(input_file) as workbook:
    data = []
    for sheet_index in range(workbook.nsheets):
         if sheet_index in my_sheets:
              worksheet = workbook.sheet_by_index(sheet_index)
              if first_worksheet:
                   header_row = worksheet.row_values(0)
                   data.append(header_row)
                   first_worksheet = False
              for row_index in range(1,worksheet.nrows):
                   row_list = []
                   sale_amount = worksheet.cell_value\
                   (row_index, sales_column_index)
                   if sale_amount > threshold:
                        for column_index in range(worksheet.ncols):
                             cell_value = worksheet.cell_value\
                             (row_index,column_index)
                             cell_type = worksheet.cell_type\
```

```
32                              (row_index, column_index)
33                              if cell_type == 3:
34                                   date_cell = xldate_as_tuple\
35                                   (cell_value,workbook.datemode)
36                                   date_cell = date(*date_cell[0:3])\
37                                   .strftime('%m/%d/%Y')
38                                   row_list.append(date_cell)
39                              else:
40                                   row_list.append(cell_value)
41                    if row_list:
42                         data.append(row_list)
43    for list_index, output_list in enumerate(data):
44         for element_index, element in enumerate(output_list):
45               output_worksheet.write(list_index, element_index, element)
46 output_workbook.save(output_file)
```

第 10 行代码创建了一个列表变量 `my_sheets`，其中包含两个整数，表示要处理的工作表的索引值。

第 16 行代码创建了工作簿中所有工作表的索引值，并在这些索引值上应用一个 `for` 循环。

第 17 行代码检验 `for` 循环中要处理的索引值是否是 `my_sheets` 中的一个索引值。这个检验确保代码只处理那些我们想处理的工作表。

因为我们在工作表索引值之间迭代，所以在第 18 行代码中，需要使用工作簿的 `sheet_by_index` 函数与索引值一起引用当前工作表。

对于要处理的第一个工作表，第 19 行代码为 `True`，所以我们将标题行追加到 `data` 中，然后将 `first_worksheet` 设为 `False`。此后，和前面的示例一样，以同样的方法处理余下的数据行。对于第二个和此后要处理的工作表，脚本直接转到第 23 行代码来处理工作表中的数据行。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 11excel_value_meets_condition_set_of_worksheets.py sales_2013.xlsx\
output_files\11output.xls
```

你可以打开输出文件 11output.xls 查看一下结果。

2. pandas

使用 pandas 在工作簿中选择一组工作表非常容易。你只需在 `read_excel` 函数中将工作表的索引值或名称设置成一个列表就可以了。在这个示例中，我们创建一个索引值列表 `my_sheets`，然后在 `read_excel` 函数中设定 `sheetname` 等于 `my_sheets`。

要使用 pandas 选择一组工作表，在文本编辑器中输入下列代码，然后将文件保存为 pandas_value_meets_condition_set_of_worksheets.py：

```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
my_sheets = [0,1]
```

```
threshold = 1900.0
data_frame = pd.read_excel(input_file, sheetname=my_sheets, index_col=None)
row_list = []
for worksheet_name, data in data_frame.items():
    row_list.append(data[data['Sale Amount'].astype(float) > threshold])
filtered_rows = pd.concat(row_list, axis=0, ignore_index=True)
writer = pd.ExcelWriter(output_file)
filtered_rows.to_excel(writer, sheet_name='set_of_worksheets', index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_value_meets_condition_set_of_worksheets.py\
sales_2013.xlsx output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.5 处理多个工作簿

本章前面的几小节演示了如何为单个工作表、工作簿中所有的工作表和工作簿中的一组工作表筛选出特定的行与特定的列。这些处理工作簿的技术是非常有用的。但是，有时你需要处理多个工作簿。在这种情况下，Python 会给你惊喜，因为它可以让你自动化和规模化地进行数据处理，远远超过手工处理能够达到的限度。

这一节重新引入了 Python 内置的 `glob` 模块，之前第 2 章中曾介绍过这个模块。在本章前面几个示例的基础上，下面演示一下如何处理多个工作簿。

为了使用多个工作簿，首先需要创建多个工作簿。那么让我们再创建另外两个 Excel 工作簿，这样就一共有 3 个工作簿了。但是，请记住这里介绍的技术可以扩展为处理任意多的文件，只要计算机能力允许。

先从下面这个步骤开始。

(1) 打开现有的工作簿 sales_2013.xlsx。

现在，创建第二个工作簿。

(2) 将现有的 3 个工作表名称改为 january_2014、february_2014 和 march_2014。
(3) 在 3 个工作表中，将 Purchase Date 列中的年份改成 2014。

每个工作表中有 6 行数据，所以你一共需要进行 18 次修改（6 行 *3 个工作表）。除了修改年份以外，不需要修改其他内容。

(4) 将第二个工作簿保存为 sales_2014.xlsx。

图 3-10 展示了修改过日期后的 january_2014 工作表中的内容。

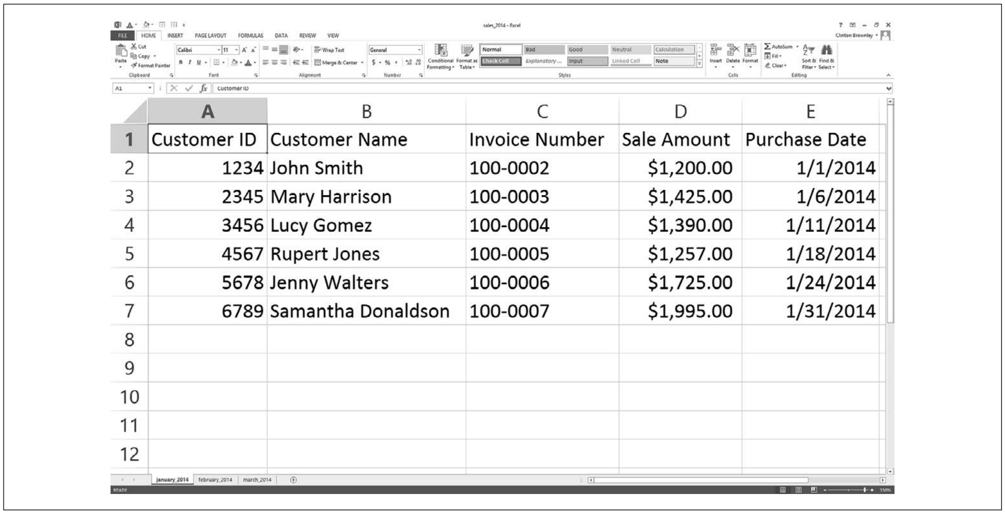


图 3-10：修改第一个工作簿中的数据，创建第二个工作簿

现在，创建第三个工作簿。

(5) 将现有的 3 个工作表名称改为 january_2015、february_2015 和 march_2015。

(6) 在 3 个工作表中，将 Purchase Date 列中的年份改成 2015。

每个工作表中有 6 行数据，所以你一共需要进行 18 次修改（6 行 *3 个工作表）。除了修改年份以外，不需要修改其他内容。

(7) 将第三个工作簿保存为 sales_2015.xlsx。

图 3-11 展示了修改过日期后的 january_2015 工作表中的内容。

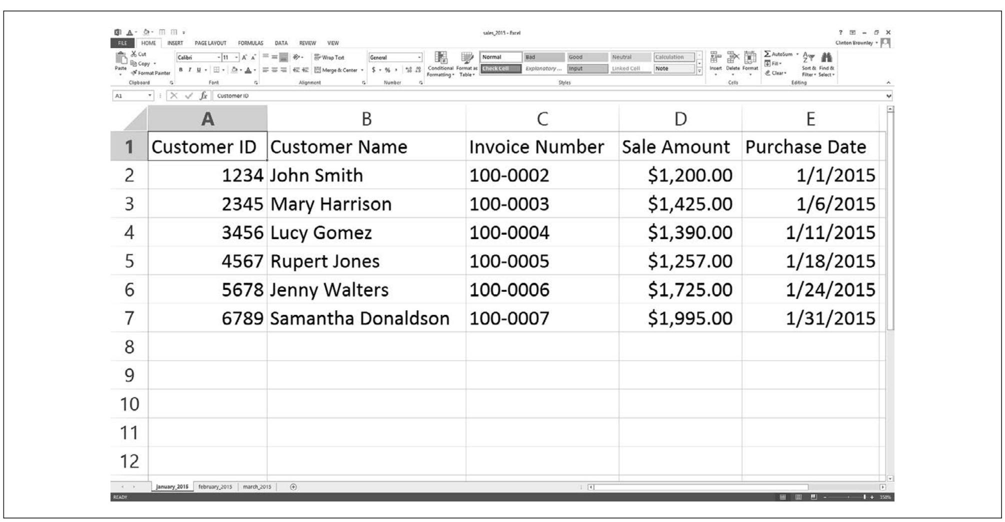


图 3-11：修改第二个工作簿中的数据，创建第三个工作簿

3.5.1　工作表计数以及每个工作表中的行列计数

在某些情况下，你知道要处理的工作簿中的内容。但是，有些时候工作簿不是你创建的，所以你不清楚其中的内容。与 CSV 文件不同，Excel 工作簿可以包含多个工作表，所以如果你不清楚这些工作表中的内容，那么在开始处理工作表之前，获取一些关于工作表的描述性信息则是非常重要的。

如果想知道一个文件夹中工作簿的数量，每个工作簿中工作表的数量，以及每个工作表中行与列的数量，在文本编辑器中输入下列代码，然后将文件保存为 12excel_introspect_all_workbooks.py：

```
 1 #!/usr/bin/env python3
 2 import glob
 3 import os
 4 import sys
 5 from xlrd import open_workbook
 6 input_directory = sys.argv[1]
 7 workbook_counter = 0
 8 for input_file in glob.glob(os.path.join(input_directory, '*.xls*')):
 9     workbook = open_workbook(input_file)
10     print('Workbook: %s' % os.path.basename(input_file))
11     print('Number of worksheets: %d' % workbook.nsheets)
12     for worksheet in workbook.sheets():
13          print('Worksheet name:', worksheet.name, '\tRows:',\
14                    worksheet.nrows, '\tColumns:', worksheet.ncols)
15     workbook_counter += 1
16 print('Number of Excel workbooks: %d' % (workbook_counter))
```

第 2 和 3 行代码分别导入 Python 内置的 `glob` 模块和 `os` 模块，以使我们可以使用其中的函数识别和解析待处理文件的路径名。

第 8 行代码使用 Python 内置的 `glob` 模块和 `os` 模块创建了一个要处理的输入文件列表，并对这个输入文件列表应用 `for` 循环，这行代码可以使我们对所有要处理的工作簿进行迭代。

第 10~14 行代码在屏幕上打印出每个工作簿的信息。第 10 行代码打印工作簿的名称。第 11 行代码打印工作簿中工作表的数量。第 13 和 14 行代码打印出工作簿中工作表的名称和每个工作表中行与列的数量。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 12excel_introspect_all_workbooks.py "C:\Users\Clinton\Desktop"
```

你应该可以看到输出被打印到屏幕上，如图 3-12 所示。

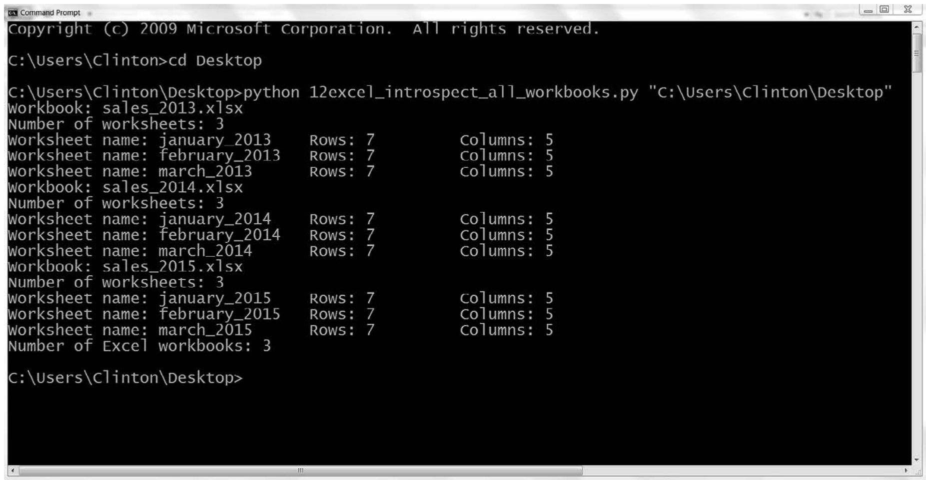


图 3-12：处理多个工作簿的 Python 脚本的输出

输出显示，脚本处理了 3 个工作簿，还打印出了 3 个工作簿的名称（例如：sales_2013.xls）、每个工作簿中 3 个工作表的名称（例如：january_2013），以及每个工作表中行与列的数量（例如：7 行和 5 列）。

当你对要处理的文件不太熟悉的时候，打印出文件的一些描述性信息是非常有用的。知道了文件的数量以及每个文件中行与列的数量，你就可以大致了解文件处理任务量和文件内容的一致性了。

3.5.2 从多个工作簿中连接数据

1. 基础Python

要使用基础 Python 将多个工作簿中所有工作表的数据垂直连接成一个输出文件，在文本编辑器中输入下列代码，然后将文件保存为 13excel_concat_data_from_multiple_workbook.py：

```
#!/usr/bin/env python3
import glob
import os
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_folder = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('all_data_all_workbooks')
data = []
first_worksheet = True
for input_file in glob.glob(os.path.join(input_folder, '*.xls*')):
    print os.path.basename(input_file)
    with open_workbook(input_file) as workbook:
```

```
17         for worksheet in workbook.sheets():
18             if first_worksheet:
19                 header_row = worksheet.row_values(0)
20                 data.append(header_row)
21                 first_worksheet = False
22             for row_index in range(1,worksheet.nrows):
23                 row_list = []
24                 for column_index in range(worksheet.ncols):
25                     cell_value = worksheet.cell_value\
26                     (row_index,column_index)
27                     cell_type = worksheet.cell_type\
28                     (row_index, column_index)
29                     if cell_type == 3:
30                         date_cell = xldate_as_tuple\
31                         (cell_value,workbook.datemode)
32                         date_cell = date(*date_cell[0:3])\
33                         .strftime('%m/%d/%Y')
34                         row_list.append(date_cell)
35                     else:
36                         row_list.append(cell_value)
37                 data.append(row_list)
38 for list_index, output_list in enumerate(data):
39     for element_index, element in enumerate(output_list):
40             output_worksheet.write(list_index, element_index, element)
41 output_workbook.save(output_file)
```

第 13 行代码创建了一个布尔型（就是 `True/False`）变量 `first_worksheet`，用来区别要处理的第一个工作表和其他后续工作表。对于要处理的第一个工作表，第 18 行代码为 `True`，所以我们将标题行追加到 `data` 中，然后将 `first_worksheet` 设为 `False`。

对于第一个工作表中余下的数据行和后续工作表中的所有行，我们跳过标题行，处理数据行。因为第 22 行代码中的 `range` 函数不是从 0 开始，而是从 1 开始的，所以我们知道是从第二行开始处理的。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 13excel_ concat_data_from_multiple_workbooks.py "C:\Users\Clinton\Desktop"\
output_files\13output.xls
```

你可以打开输出文件 13output.xls 查看一下结果。

2. pandas

pandas 提供了 `concat` 函数来连接数据框。如果你想把数据框一个一个地垂直堆叠起来，那么就要设置参数 `axis=0`。如果你想把数据框一个一个地平行连接起来，那么就要设置参数 `axis=1`。此外，如果你需要基于某个关键字列连接数据框，pandas 中的 `merge` 函数可以提供类似 SQL join 的操作（如果你不理解这个，没有关系，接下来的第 4 章中会有更多关于数据库的介绍）。

要使用 pandas 将多个工作簿中所有工作表的数据垂直连接成一个输出文件，在文本编辑器中输入下列代码，然后将文件保存为 pandas_concat_data_from_multiple_workbook.py：

```
#!/usr/bin/env python3
```

```
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_workbooks = glob.glob(os.path.join(input_path,'*.xls*'))
data_frames = []
for workbook in all_workbooks:
    all_worksheets = pd.read_excel(workbook, sheetname=None, index_col=None)
    for worksheet_name, data in all_worksheets.items():
         data_frames.append(data)
all_data_concatenated = pd.concat(data_frames, axis=0, ignore_index=True)
writer = pd.ExcelWriter(output_file)
all_data_concatenated.to_excel(writer, sheet_name='all_data_all_workbooks',\
index=False)
writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_concat_data_from_multiple_workbooks.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

3.5.3　为每个工作簿和工作表计算总数和均值

1. 基础Python

要使用基础 Python 为多个工作簿计算工作表级别和工作簿级别的统计量，在文本编辑器中输入下列代码，然后将文件保存为 14excel_sum_average_multiple_workbook.py：

```
#!/usr/bin/env python3
import glob
import os
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
from xlwt import Workbook
input_folder = sys.argv[1]
output_file = sys.argv[2]
output_workbook = Workbook()
output_worksheet = output_workbook.add_sheet('sums_and_averages')
all_data = []
sales_column_index = 3
header = ['workbook', 'worksheet', 'worksheet_total', 'worksheet_average',\
                        'workbook_total', 'workbook_average']
all_data.append(header)
for input_file in glob.glob(os.path.join(input_folder, '*.xls*')):
    with open_workbook(input_file) as workbook:
         list_of_totals = []
         list_of_numbers = []
         workbook_output = []
         for worksheet in workbook.sheets():
              total_sales = 0
```

```
24            number_of_sales = 0
25            worksheet_list = []
26            worksheet_list.append(os.path.basename(input_file))
27            worksheet_list.append(worksheet.name)
28            for row_index in range(1,worksheet.nrows):
29                try:
30                    total_sales += float(str(worksheet.cell_value\
31                    (row_index,sales_column_index))\
32                    .strip('$').replace(',',''))
33                    number_of_sales += 1.
34                except:
35                    total_sales += 0.
36                    number_of_sales += 0.
37            average_sales = '%.2f' % (total_sales / number_of_sales)
38            worksheet_list.append(total_sales)
39            worksheet_list.append(float(average_sales))
40            list_of_totals.append(total_sales)
41            list_of_numbers.append(float(number_of_sales))
42            workbook_output.append(worksheet_list)
43        workbook_total = sum(list_of_totals)
44        workbook_average = sum(list_of_totals)/sum(list_of_numbers)
45        for list_element in workbook_output:
46            list_element.append(workbook_total)
47            list_element.append(workbook_average)
48        all_data.extend(workbook_output)
49
50 for list_index, output_list in enumerate(all_data):
51     for element_index, element in enumerate(output_list):
52         output_worksheet.write(list_index, element_index, element)
53 output_workbook.save(output_file)
```

第 12 行代码创建了一个空列表 `all_data`，用来保存要写入输出文件的所有行。第 13 行代码创建了一个变量 `sales_column_index`，保存 Sale Amount 列的索引值。

第 14 行代码为输出文件创建了一个列标题列表，并使用第 16 行代码将其追加到 `all_data` 中。

在第 19、20 和 21 行代码中，分别创建了 3 个列表。`list_of_totals` 用来保存工作簿中所有工作表的销售额总计。同样，`list_of_numbers` 用来保存工作簿的所有工作表中用来计算总销售额的销售额数据个数。第三个列表，`workbook_output`，用来保存要写入输出文件的所有输出列表。

第 25 行代码创建了一个列表 `worksheet_list`，用来保存要保留的所有工作表的信息。在第 26 和 27 行代码中，将工作簿名称和工作表名称追加到 `worksheet_list` 中。同样，在第 38 和 39 行代码中，将销售额总计和均值追加到 `worksheet_list` 中。在第 42 行代码中，将 `worksheet_list` 追加到 `workbook_output` 中，在工作簿级别保存信息。

在第 40 和 41 行代码中，将工作表的销售额总计和销售额数据个数分别追加到 `list_of_totals` 和 `list_of_numbers` 中，这样我们可以对所有工作表保存这些值。在第 43 和 44 行代码中，使用这两个列表计算出工作簿的销售额总计和销售额均值。

在第 45~47 行代码中，我们在 `workbook_output` 的各个列表之间迭代（每个工作簿有 3 个

列表，因为每个工作簿有 3 个工作表），并将工作簿级别的销售额总计和均值追加到每个列表中。

当获得了所有要为工作簿保留的信息之后（就是 3 个列表，每个工作表有一个列表），就将这些列表扩展到 all_data 中。我们使用 extend，不是 append，以使 workbook_output 中的每个列表都会成为 all_data 中的一个独立元素。这样的话，在处理完所有工作簿之后，all_data 就是一个具有 9 个元素的列表，每个元素都是一个列表。否则，如果使用 append，all_data 中就会只有 3 个元素，每个元素都是一个列表的列表。

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python 14excel_sum_average_multiple_workbooks.py "C:\Users\Clinton\Desktop"\
output_files\14output.xls
```

你可以打开输出文件 14output.xls 查看一下结果。

2. pandas

pandas 可以直接在多个工作簿之间迭代，并可以同时在工作簿级别和工作表级别计算统计量。在下面的脚本中，为工作簿中的每个工作表计算统计量，然后将结果连接成一个数据框。接下来，计算工作簿级别的统计量，将它们转换成一个数据框，然后通过基于工作簿名称的左连接将两个数据框合并在一起，并将结果数据框添加到一个列表中。当所有工作簿级别的数据框都进入列表之后，将这些数据框连接成一个独立数据框，并写入输出文件。

要使用 pandas 计算工作表级别和工作簿级别的统计量，在文本编辑器中输入下列代码，然后将文件保存为 pandas_sum_average_multiple_workbook.py：

```
#!/usr/bin/env python3
import pandas as pd
import glob
import os
import sys
input_path = sys.argv[1]
output_file = sys.argv[2]
all_workbooks = glob.glob(os.path.join(input_path,'*.xls*'))
data_frames = []
for workbook in all_workbooks:
    all_worksheets = pd.read_excel(workbook, sheetname=None, index_col=None)
    workbook_total_sales = []
    workbook_number_of_sales = []
    worksheet_data_frames = []
    worksheets_data_frame = None
    workbook_data_frame = None
    for worksheet_name, data in all_worksheets.items():
         total_sales = pd.DataFrame([float(str(value).strip('$').replace(\
         ',',''))
         for value in data.loc[:, 'Sale Amount']]).sum()
         number_of_sales = len(data.loc[:, 'Sale Amount'])
         average_sales = pd.DataFrame(total_sales / number_of_sales)

         workbook_total_sales.append(total_sales)
         workbook_number_of_sales.append(number_of_sales)
```

```
            data = {'workbook': os.path.basename(workbook),
                    'worksheet': worksheet_name,
                    'worksheet_total': total_sales,
                    'worksheet_average': average_sales}

            worksheet_data_frames.append(pd.DataFrame(data, \
            columns=['workbook', 'worksheet', \
            'worksheet_total', 'worksheet_average']))
        worksheets_data_frame = pd.concat(\
        worksheet_data_frames, axis=0, ignore_index=True)
        workbook_total = pd.DataFrame(workbook_total_sales).sum()
        workbook_total_number_of_sales = pd.DataFrame(\
        workbook_number_of_sales).sum()
        workbook_average = pd.DataFrame(\
        workbook_total / workbook_total_number_of_sales)

        workbook_stats = {'workbook': os.path.basename(workbook),
                          'workbook_total': workbook_total,
                          'workbook_average': workbook_average}
        workbook_stats = pd.DataFrame(workbook_stats, columns=\
        ['workbook', 'workbook_total', 'workbook_average'])
        workbook_data_frame = pd.merge(worksheets_data_frame, workbook_stats, \
        on='workbook', how='left')
        data_frames.append(workbook_data_frame)
    all_data_concatenated = pd.concat(data_frames, axis=0, ignore_index=True)
    writer = pd.ExcelWriter(output_file)
    all_data_concatenated.to_excel(writer, sheet_name='sums_and_averages', \
    index=False)
    writer.save()
```

要运行这个脚本，在命令行中输入以下命令，然后按回车键：

```
python pandas_sum_average_multiple_workbooks.py "C:\Users\Clinton\Desktop"\
output_files\pandas_output.xls
```

你可以打开输出文件 pandas_output.xls 查看一下结果。

本章介绍了很多基础性操作，包括读取和分析 Excel 工作簿、在工作表中浏览行与列、处理多个 Excel 工作表、处理多个 Excel 工作簿、以及为多个 Excel 工作表和工作簿计算统计量的方法。如果一直跟随本章内容练习示例代码，你应该完成了 14 个新的 Python 脚本！

练习本章中示例代码的最大收获是，你很好地掌握了浏览和处理 Excel 文件的技术，而 Excel 文件是商业过程中最常用的一种文件。而且，因为很多商业机构将数据保存在 Excel 工作簿中，所以你现在已经掌握了处理这些工作簿中数据的一系列方法，无论工作簿的数量和体积有多大，也无论每个工作簿中有多少工作表，你都可以使用计算机数据处理的强大能力来自动化和规模化地处理和分析 Excel 工作簿中的各种数据。

我们要面对的下一个数据源是数据库。因为数据库是一种非常常用的数据存储，所以知道如何访问其中的数据是非常重要的。只要知道了如何访问其中的数据，你就可以像处理 CSV 文件和 Excel 文件一样，同样以一行接一行的方式来处理这些数据。掌握了第 2 章和本章中的示例代码之后，你已经完全做好了处理数据库中数据的准备。

3.6 本章练习

(1) 对根据具体条件、集合和正则表达式来筛选行数据的一个脚本进行修改，将与示例代码中不同的一组数据打印出来并写入输出文件。

(2) 对根据索引值或列标题来筛选列数据的一个脚本进行修改，将与示例代码中不同的一组数据打印出来并写入输出文件。

(3) 创建一个新的 Python 脚本，将筛选行或列的某个脚本中的代码与从多个工作簿中连接数据的某个脚本中的代码组合起来，生成一个新的输出文件，其中包含来自于多个工作簿的特定行与特定列。

第 4 章

数据库

与电子表格一样，数据库在商业中的应用也非常普遍。公司使用数据库保存客户、库存和雇员数据。在对运营、销售和财务等活动的跟踪方面，数据库也至关重要。与简单电子表格或工作簿电子表格不同，数据库中的表是互相关联的，就像一个电子表格中的一行可以和另一个电子表格中的一行或一列关联起来一样。我们给出一个标准的例子，如客户数据（姓名、地址等）可以（通过客户 ID 编号）关联到“订单”电子表格中的一行，其中包含预订的物品。这些物品又可以关联到“供应商”电子表格中的数据，使你不但可以跟踪并完成订货，还可以进行更深层次的分析。虽然你可以使用 Python 对 CSV 文件和 Excel 文件这些既常用又重要的数据源进行自动化和规模化的处理，而且掌握了处理这些文件的技能不论从学习的角度（可以学习通用的编程操作）还是从实用的角度（大量商业数据保存在这些类型的文件中）都非常重要，但是，使用数据库却可以将计算机完成任务的能力提高成千上万倍。

> **关系数据库**
>
> 本章将要讨论**关系数据库**（relational database）和**关系数据库管理系统**（RDBMS）。在关系数据库中，保存信息的表由表间定义好的关系相关联，例如，可以使用像“订单 ID 编号”这样的关键字将客户记录与产品记录、运输记录等关联起来。在某些情况下（通常是“大数据”情况下），定义所有的关系对运营来说没必要，或者需要太多的计算能力。这样就出现了**非关系型数据库**（non-relational database），它以其他方式存储并搜索数据。举例来说，它不会将位于不同表中的客户记录和订单记录关联起来，而是将所有订单顺序存储在一个记录中，并将客户数据作为订单数据的一个子集。（在这种情况下，你可以节省掉在另一个表中寻找客户数据的开销，但付出的代价是，每次客户新增一个订单时，都要再保存一份客户数据副本。）本书虽然不涉及非关系型数据库，但你应该知道，（a）它们确实存在而且（b）有可以处理很多种非关系型数据库中数据的 Python 模块。

要学习如何使用 Python 同数据库交互，首先你要有个数据库，并且数据库中要有一张充满了数据的表。如果你还没有使用过这样的数据库和数据表，那么这就是你需要解决的主要问题。幸运的是，有两种资源可供选择，它们可以让我们轻松愉快地使用本章中的示例开始学习。

首先，Python 有个内置模块 `sqlite3`，它可以使我们创建内存数据库。这就是说我们可以使用 Python 代码直接创建一个数据库和其中充满数据的表，不用下载安装专门的数据库软件。在本章的前半部分，我们就使用这个功能快速地开始本章的学习，并把重点放在与数据库、表和数据的交互上面，而不用考虑如何下载和安装数据库。

其次，你可能已经使用过 MySQL、PostgreSQL 或 Oracle 这样的常用数据库系统。这些数据库系统的开发公司已经使这些系统非常易于下载和安装了。尽管你可能不会每天都使用数据库系统，但是它们在商业中的应用却非常普遍。因此，熟悉一些常用的数据库操作，并知道如何使用 Python 来完成这些操作对你来说是非常重要的。在本章的后半部分，我们要下载并安装一个数据库系统，这样你就可以使用在本章前半部分学到的知识，与一个实际数据库系统进行方便地交互并操作其中的数据了。

什么是 SQL

你会发现本章使用的多数模块和软件的名称中都有“SQL”。SQL（通常读作“sequel”，尽管有人坚持读作“es-queue-el”）表示**结构化查询语言**（Structured Query Language），是一组应用广泛的与数据库进行交互的命令。SQL 的版本很多，你的数据库系统也可能使用专门的命令和语法，但某些确定的操作比如 `SELECT`、`JOIN`、`INSERT` 和 `UPDATE` 对所有版本都是通用的。本章会教你一些基础知识，包括完全使用 Python 建立一个数据库，以及使用 SQL 从数据库中将数据输送到 Python 代码中以供处理。

4.1 Python内置的sqlite3模块

正如上面所提到的，我们使用 Python 内置的 `sqlite3` 模块直接在 Python 代码中创建一个内存数据库以及充满了数据的表，从而快速开始本章的学习。与第 2 章和第 3 章一样，第一个示例的重点在于演示如何对 SQL 查询输出的行进行计数。在任何不确定你的查询会输出多少行的情况下，这个功能非常重要。通过这个功能，在开始工作以前，你可以知道有多少行数据需要处理。这个示例还很实用，因为我们会使用很多在 Python 中与数据库交互相关联的语法，用来创建数据库中的表、在表中插入数据和从输出中获取数据并对行进行计数。你将看到很多语法会在本章的示例中多次重复出现。

现在就开始吧。要创建数据库中的表、在表中插入数据，以及在输出中获取数据并对行进行计数，在文本编辑器中输入下列代码，然后将文件保存为 1db_count_rows.py：

```
#!/usr/bin/env python3
import sqlite3

# 创建SQLite3内存数据库
# 创建带有4个属性的sales表
```

```
con = sqlite3.connect(':memory:')
query = """CREATE TABLE sales
            (customer VARCHAR(20),
            product VARCHAR(40),
            amount FLOAT,
            date DATE);"""
con.execute(query)
con.commit()

# 在表中插入几行数据
data = [('Richard Lucas', 'Notepad', 2.50, '2014-01-02'),
      ('Jenny Kim', 'Binder', 4.15, '2014-01-15'),
      ('Svetlana Crow', 'Printer', 155.75, '2014-02-03'),
      ('Stephen Randolph', 'Computer', 679.40, '2014-02-20')]
statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
con.executemany(statement, data)
con.commit()

# 查询sales表
cursor = con.execute("SELECT * FROM sales")
rows = cursor.fetchall()

# 计算查询结果中行的数量
row_counter = 0
for row in rows:
    print(row)
    row_counter += 1
print('Number of rows: %d' % (row_counter))
```

图 4-1、图 4-2 和图 4-3 分别展示了在 Anaconda Spyder、Notepad++（Windows）和 TextWrangler（macOS）中编辑脚本的界面。

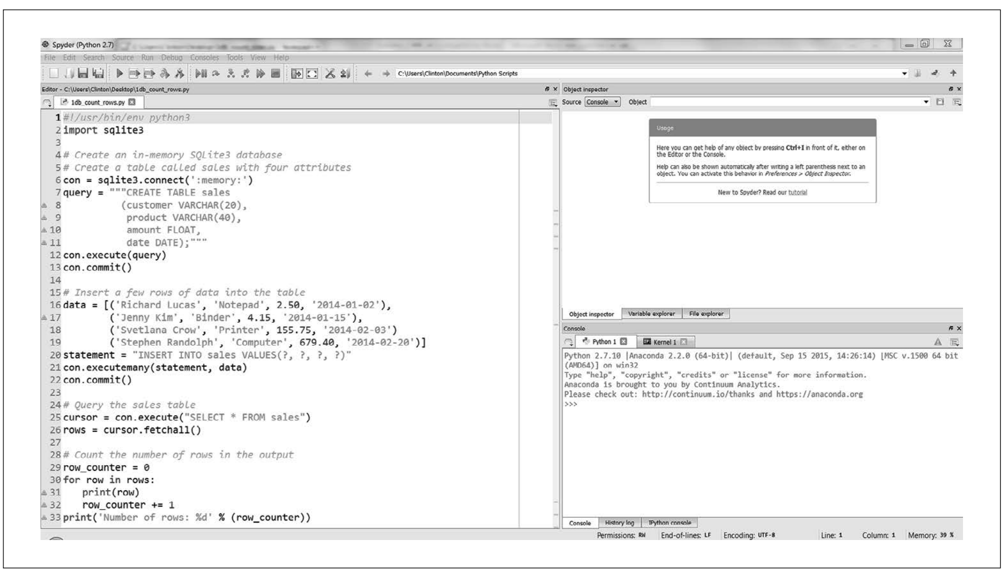


图 4-1：Anaconda Spyder 中的 Python 脚本 1db_count_rows.py

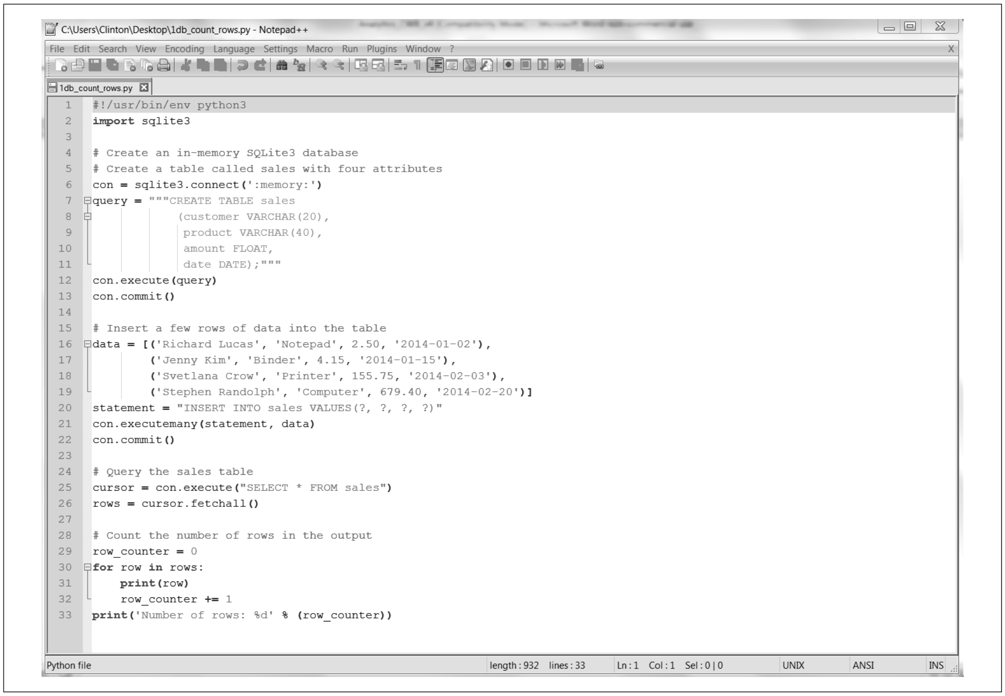


图 4-2：Notepad++（Windows）中的 Python 脚本 1db_count_rows.py

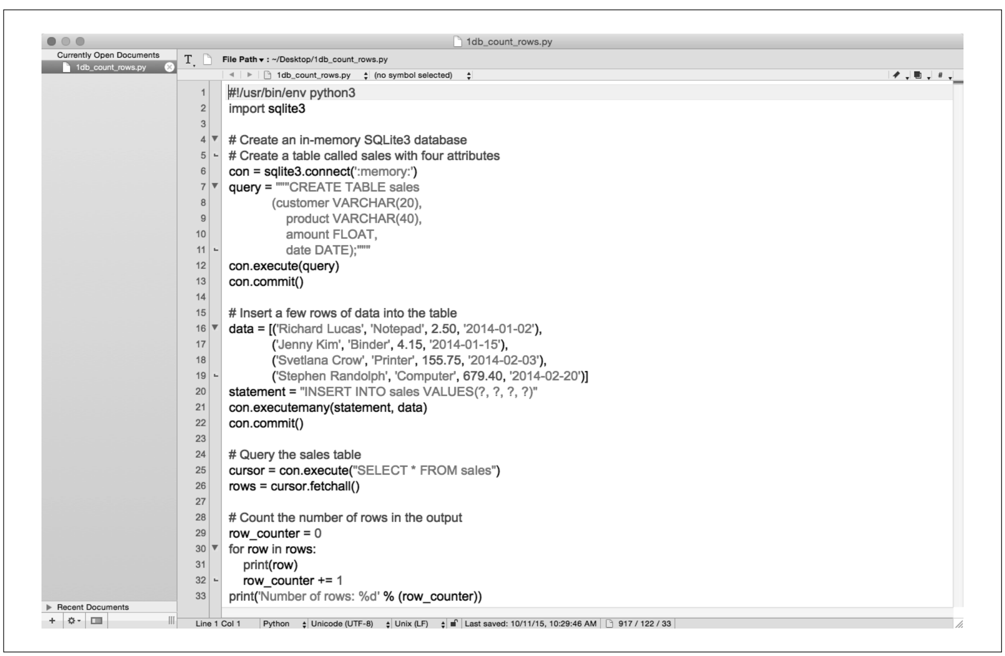


图 4-3：TextWrangler（macOS）中的 Python 脚本 1db_count_rows.py

在这些图中应该可以看出，与处理 CSV 文件和 Excel 工作簿不同，要与数据库进行交互，还需要学习一些新的语法。

第 2 行代码导入 sqlite3 模块，它提供了一个轻量级的基于磁盘的数据库，不需要独立的服务器进程，并且允许使用 SQL 查询语言的一个变种去访问数据库。SQL 命令在本章的示例代码中均使用大写字母表示。因为本章是关于在 Python 中与数据库进行交互的，所以这里主要介绍了通用的 CRUD（也就是 Create、Read、Update 和 Delete，即创建、读取、更新和删除）数据库操作[1]。示例代码包括创建数据库和表（Create）、向表中插入数据（Create）、更新表中数据（Update）和从表中选择特定的行（Read）。这些 SQL 操作在各个关系数据库中是通用的。

为了使用这个模块，首先必须创建一个代表数据库的连接对象。第 6 行代码创建了连接对象 con 来代表数据库。在这个示例中，我使用专用名称 ':memory:' 在内存中创建了一个数据库。如果你想要这个数据库持久化，就需要提供另外的字符串。例如，如果我使用字符串 'My_database.db' 或 'C:\Users\Clinton\Desktop\my_database.db'，而不是 ':memory:'，那么数据库对象就会持久保存在当前目录或你的桌面上。

第 7~11 行代码使用 3 个双引号创建了一个多行字符串，并将这个字符串赋给变量 query。这个字符串是一个 SQL 命令，可以在数据库中创建一个名为 sales 的表。sales 表有 4 个属性：customer、product、amount 和 date。customer 属性是一个变长字符型字段，最大字符数为 20。product 属性也是个变长字符型字段，最大字符数为 40。amount 属性是一个浮点数型字段。date 属性是一个日期型字段。

第 12 行代码使用连接对象的 execute() 方法执行包含在变量 query 中的 SQL 命令，在内存数据库中创建 sales 表。

第 13 行代码使用连接对象的 commit() 方法将修改提交（也就是保存）到数据库。当你对数据库做出修改时，必须使用 commit() 方法来保存你的修改，否则，这种修改就不会保存到数据库中。

第 16 行代码创建了一个元组列表，并将这个列表赋给变量 data。列表中的每个元素都是一个包含 4 个值的元组：3 个字符串和 1 个浮点数。这 4 个值按位置对应了表的 4 个属性（就是表中的 4 列）。还有，每个元组包含了表中的一行数据。因为列表包含 4 个元组，所以它包含了表中的 4 行数据。

第 20 行代码与第 7 行代码类似，创建了一个字符串并将其赋给变量 statement。因为这个字符串可以写在一行中，所以包含在一对双引号中，不像在第 7 行代码中，要使用一对 3 个双引号来表示多行字符串。这行代码中的字符串是另一个 SQL 命令，是一个 INSERT 语句，可以将 data 中的数据行插入 sales 表。当你第一次看到这行代码时，会很想知道问号（?）的作用。问号在这里用作占位符，表示你想在 SQL 命令中使用的值。然后，在连接对象的 execute() 或 executemany() 方法中，你需要提供一个包含 4 个值的元组，元组中的值会按位置替换到 SQL 命令中。相对于使用字符串操作组装 SQL 命令的方法，这种参

注 1：在 http://en.wikipedia.org/wiki/Create,_read,_update_and_delete 这个网址你可以学到更多关于 CRUD 操作的知识。

数替换的方法可以使你的代码不易受到 SQL 注入攻击[2]，这种攻击确实有害于我们的系统。

第 21 行代码使用连接对象的 executemany() 方法为 data 中的每个数据元组执行变量 statement 中的 SQL 命令。因为 data 中有 4 个数据元组，所以这个 executemany() 方法执行 4 次 INSERT 语句，高效率地将 4 行数据插入 sales 表。

请记住，在介绍第 13 行代码时我们强调过，当你对数据库做出修改后，必须使用 commit() 方法，否则你的修改就不会保存到数据库中。将 4 行数据插入 sales 表肯定是对数据库进行了修改，所以在第 22 行代码中，又一次使用连接对象的 commit() 方法将修改保存到数据库。

现在我们的内存数据库中有一个 sales 表，表中有 4 行数据，下面学习一下如何从数据库的表中提取数据。第 25 行代码使用连接对象的 execute() 方法运行一条 SQL 命令，并将命令结果赋给一个光标对象 cursor。光标对象有若干方法，例如，execute、executemany、fetchone、fetchmany 和 fetchall。因为经常需要查看或处理在 execute() 方法中运行的 SQL 命令的全部结果，所以我们通常使用 fetchall() 方法取出（返回）结果集中的所有行。

第 26 行代码执行了 fetchall() 方法，使用光标对象的这个方法返回在第 25 行代码中执行的 SQL 命令的结果集中的所有行，并将这些行赋给列表变量 rows。这样，变量 rows 就是包含了所有来自于第 25 行代码中的 SQL 命令的数据行的列表。每行数据都是一个元组，所以 rows 是一个元组列表。在这种情况下，因为我们已知 sales 表中包含 4 行数据，并且 SQL 命令从 sales 表中选择所有的行，所以 rows 是一个含有 4 个元组的列表。

最后，在第 29~33 行代码中，又回到了熟悉的基础操作，创建了一个变量 row_counter 来计算 rows 中行的数量，创建了一个 for 循环在 rows 的行中迭代，对于 rows 中的每一行，使 row_counter 增加 1，结果，当 for 循环结束了在 rows 的所有行中的迭代之后，在命令行窗口（或终端窗口）中打印出字符串 Number of rows：和 row_counter 中的值。正如之前说过的，我们希望 rows 中有 4 行数据。

要看看这个 Python 脚本的实际运行情况，根据不同操作系统，在命令行中输入下面的一个命令，然后按回车键。

- Windows 操作系统

```
python 1db_count_rows.py
```

- macOS 操作系统

```
chmod +x 1db_count_rows.py
./1db_count_rows.py
```

你可以看到输出被打印到屏幕上，如图 4-4（Windows 系统）或图 4-5（macOS 系统）所示。

注 2：SQL 注入攻击是一种恶意 SQL 语句，攻击者使用它来获取私人信息或损坏数据存储与应用程序。要获取更多关于 SQL 注入攻击的信息，请参考 http://en.wikipedia.org/wiki/SQL_injection。

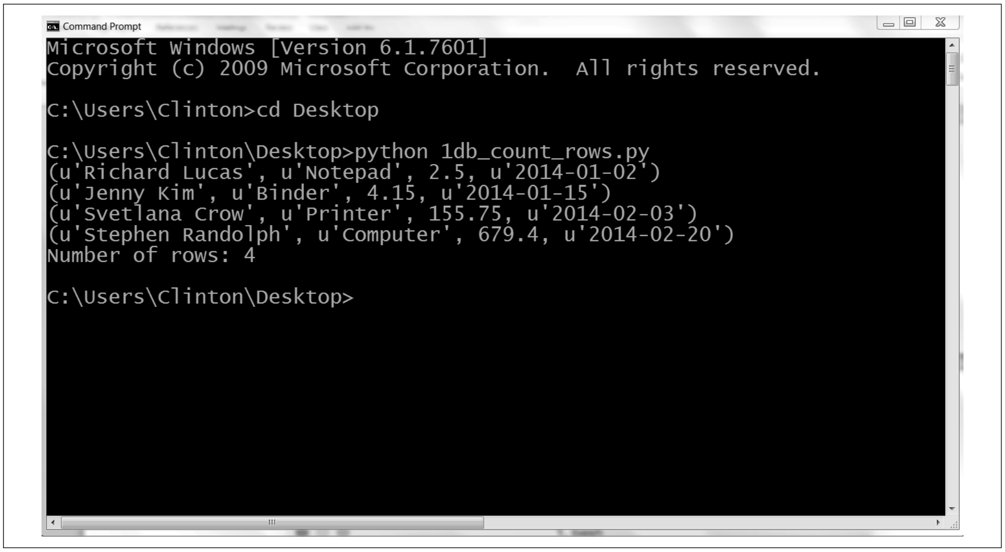


图 4-4：Windows 系统上 1db_count_rows.py 的输出，创建 SQLite3 数据库和表，向表中插入 4 行数据，在表中查询所有数据，并将结果打印在屏幕上

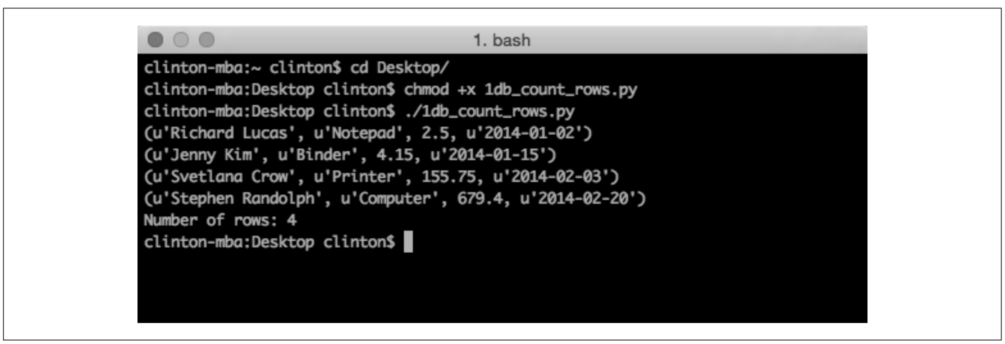


图 4-5：macOS 系统上 1db_count_rows.py 的输出

输出显示，`sales` 表中有 4 条记录。扩展一下，输出还可以显示我们创建了内存数据库、创建了表 `sales`、在表中添加了 4 条数据、从表中取出所有行，以及计算输出行的数量。

现在你已经掌握了一些基础操作，包括创建内存数据库、创建表、向表中加载数据，以及从表中读取数据。下面学习如何使用 CSV 文件向表中添加数据和更新表中数据，扩展一下你的数据库操作能力。

4.1.1　向表中插入新记录

前面的示例代码介绍了向表中加载数据的基本操作，但是这个示例有个严重的局限性，就是需要将要加载到表中的数据手写到代码中。如果我们要向表中加载 10 000 条记录，每条记录有 20~30 个字段，那将如何处理？无需多说，手动输入数据不具有扩展性。

在很多情况下，需要加载到数据表中的数据或者是一个数据库查询的结果，或者是保存在一个或多个 Excel 文件或 CSV 文件中的数据。因为对于多数数据库系统来说，将一个数据库查询结果导出为 CSV 文件非常容易，并且我们已经学习了如何处理 Excel 文件和 CSV 文件，所以现在再学习另外一种数据加载方式，也就是如何从 CSV 格式的输入文件将数据批量地**加载**到数据库的表中。

让我们创建一个新的 Python 脚本。这个脚本将创建一个数据表，向表中插入 CSV 文件中的数据，然后展示表中的数据。其中的第三个步骤，即在命令行窗口或终端窗口打印数据，不是必须的（而且如果你加载了几千条记录的话，我也不建议将记录打印在窗口中），我之所以在示例中包括了这个步骤，是想演示一种不指定单独的列索引，打印出每条记录中所有列的方法（也就是说，这种语法可以扩展到任意数目的列）。下面开始操作，在文本编辑器中输入下列代码，然后将文件保存为 2db_insert_row.py：

```
#!/usr/bin/env python3
import csv
import sqlite3
import sys
# CSV输入文件的路径和文件名
input_file = sys.argv[1]
# 创建SQLite3内存数据库
# 创建带有5个属性的Suppliers表
con = sqlite3.connect('Suppliers.db')
c = con.cursor()
create_table = """CREATE TABLE IF NOT EXISTS Suppliers
                   (Supplier_Name VARCHAR(20),
                   Invoice_Number VARCHAR(20),
                   Part_Number VARCHAR(20),
                   Cost FLOAT,
                   Purchase_Date DATE);"""
c.execute(create_table)
con.commit()
# 读取CSV文件
# 向Suppliers表中插入数据
file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
header = next(file_reader, None)
for row in file_reader:
    data = []
    for column_index in range(len(header)):
        data.append(row[column_index])
    print(data)
    c.execute("INSERT INTO Suppliers VALUES (?, ?, ?, ?, ?);", data)
con.commit()
print('')
# 查询Suppliers表
output = c.execute("SELECT * FROM Suppliers")
rows = output.fetchall()
for row in rows:
    output = []
    for column_index in range(len(row)):
        output.append(str(row[column_index]))
    print(output)
```

这个脚本和第 2 章中的脚本一样，依赖于 csv 模块和 sys 模块。第 2 行代码导入 csv 模块，以使我们使用其中的函数读取和分析 CSV 格式的输入文件。第 4 行代码导入 sys 模块，这样我们就可以在命令行中提供一个文件的路径和名称供脚本使用。第 3 行代码导入 sqlite3 模块，我们可以使用其中的方法创建简单的本地数据库和数据表，也可以执行 SQL 查询。

第 6 行代码使用 sys 模块在命令行中读取文件的路径和名称，并将这个值赋给变量 input_file。

第 9 行代码为一个简单的本地数据库 Suppliers.db 创建连接。我提供了一个数据库名称，而不使用专门的关键字 ':memory:'，来演示如何创建一个持久化数据库，当你重启计算机时，这个数据库不会被删除。因为你要把这个脚本保存在桌面上，所以 Suppliers.db 也要保存在桌面上。如果你想将数据库保存在其他位置，可以选择一个路径，比如 'C:\Users\<*Your Name*>\Documents\Supplier.db'，用来代替 'Suppliers.db'。

第 10~18 行代码创建了一个光标，以及一个多行 SQL 语句，用来创建一个具有 5 个列属性的数据表 Suppliers。执行这条 SQL 语句，并将修改提交到数据库。

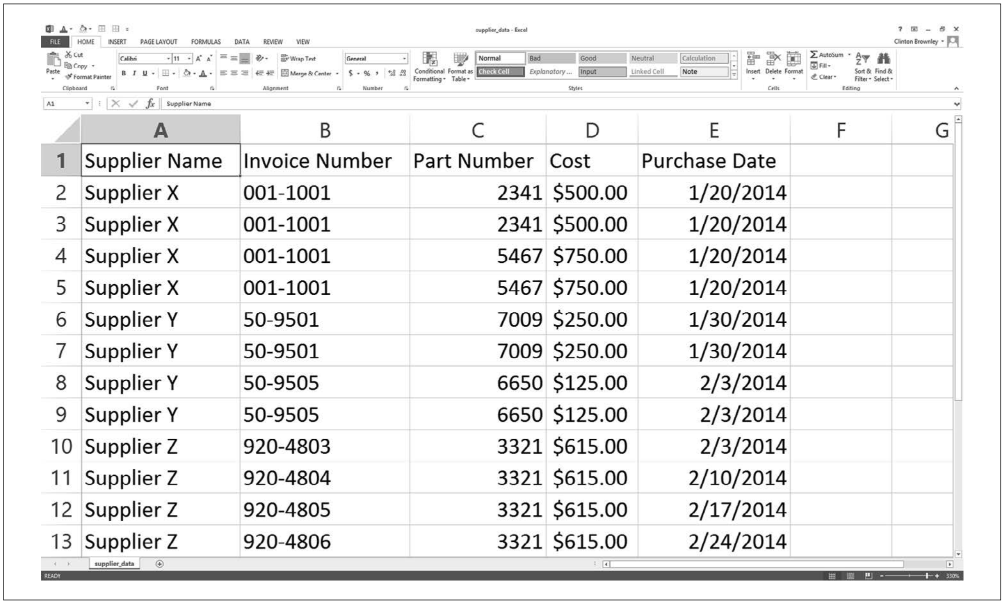
第 21~29 行代码从 CSV 格式的输入文件中读取要加载到数据库中数据，并对输入文件中的每行数据执行一条 SQL 语句，将数据插入到数据库的表中。第 21 行代码使用 csv 模块创建 file_reader 对象。第 22 行代码使用 next() 方法从输入文件中读入第一行，也就是标题行，然后将其赋给变量 header。第 23 行代码创建了一个 for 循环，在输入文件的所有数据行之间循环。第 24 行代码创建了一个空列表变量 data。对于输入中的每一行，用行中的数据去填充 data，这些数据将在第 28 行代码中的 INSERT 语句中使用。第 25 行代码创建了一个 for 循环，在每行数据的各个列之间循环。第 26 行代码通过列表的 append() 方法使用输入文件这一行中所有的数据去填充 data。第 27 行代码在命令行窗口或终端窗口中打印出追加到 data 中的这行数据。请注意缩进。第 27 行代码的缩进是在外部 for 循环之下，不是在内部 for 循环之下的，所以它是对于输入文件中的每一行来执行，而不是对于输入文件中的每一行和每一列去执行。这一行有助于调试脚本，但是一旦你确定代码能够正确运行，就可以将这行代码删除或者注释掉，这样在屏幕上就不会打印出过多的输出。

第 28 行代码是实际将每行数据加载到数据表中的代码。这行代码使用光标对象的 execute() 方法执行一条 INSERT 语句，将一行数据插入到表 Suppliers 中。问号 ? 是要插入的每个实际值的占位符。问号的数量对应着输入文件中列的数量，它们都对应着数据表中列的数量。而且，输入文件中列的顺序也要对应数据表中列的顺序。要替换到问号位置的值来自于列表 data，这个列表要放在 execute() 语句中逗号的后面。因为 data 是使用输入文件中的每行数据填充的，而且 INSERT 语句也是对于输入文件中的每行数据执行的，所以这些代码可以高效地从输入文件中读取数据行，并把这些数据行加载到数据表中。最后，第 29 行代码是另一个 commit 语句，将修改提交到数据库。

第 32~38 行代码演示了如何从数据表 Suppliers 中选择所有数据，并将输出打印到命令行窗口或者终端窗口。第 32 和 33 行代码执行一条 SQL 语句，从 Suppliers 表中选择所有数据，并将“output”中的所有行读入变量“rows”。第 34 行代码创建了一个 for 循环，在

“rows”的每一行中循环。第 36 行代码创建了一个 `for` 循环，在每行的各个列之间循环。第 37 行代码将每列中的值追加到一个名为“output”的列表中。最后，第 38 行代码中的 `print` 语句确保输出中的每一行都被打印到一个新行中（请注意缩进，是在行循环中，不是在列循环中）。

现在我们需要一个 CSV 格式的输入文件，其中包含着要加载到数据表中的所有数据。对于这个示例，可以使用在第 2 章中用过的 supplier_data.csv 文件。如果你跳过了第 2 章，或者没有这个文件，那么可以在图 4-6 中看到这个文件中的数据。



	A	B	C	D	E	F	G
1	Supplier Name	Invoice Number	Part Number	Cost	Purchase Date		
2	Supplier X	001-1001	2341	$500.00	1/20/2014		
3	Supplier X	001-1001	2341	$500.00	1/20/2014		
4	Supplier X	001-1001	5467	$750.00	1/20/2014		
5	Supplier X	001-1001	5467	$750.00	1/20/2014		
6	Supplier Y	50-9501	7009	$250.00	1/30/2014		
7	Supplier Y	50-9501	7009	$250.00	1/30/2014		
8	Supplier Y	50-9505	6650	$125.00	2/3/2014		
9	Supplier Y	50-9505	6650	$125.00	2/3/2014		
10	Supplier Z	920-4803	3321	$615.00	2/3/2014		
11	Supplier Z	920-4804	3321	$615.00	2/10/2014		
12	Supplier Z	920-4805	3321	$615.00	2/17/2014		
13	Supplier Z	920-4806	3321	$615.00	2/24/2014		

图 4-6：CSV 文件 supplier_data.csv 中的示例数据，显示在 Excel 工作表中

当你有了 Python 脚本和 CSV 输入文件之后，就可以使用脚本将 CSV 输入文件中的数据加载到 `Suppliers` 数据表中了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 2db_insert_rows.py supplier_data.csv
```

图 4-7 展示了在命令行窗口中打印输出的界面。输出的第一部分是从 CSV 文件中解析出的数据行，输出的第二部分是同样的行，只不过是从 sqlite 数据表中提取出来的。

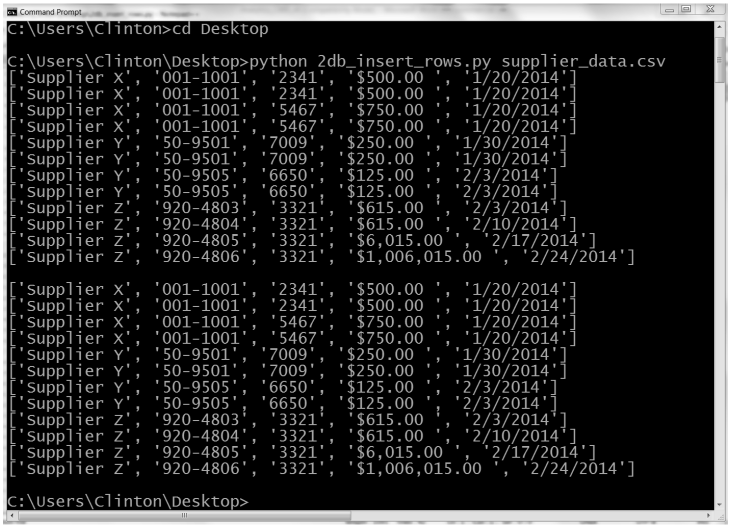


图 4-7：Windows 系统下 2db_insert_rows.py 的输出

这个输出展示了 12 个值列表，来自于 CSV 输入文件中除标题行之外的 12 行数据。在这些来自于输入文件的 12 行值列表之下，有一个空行，然后是从数据表中提取的 12 行数据的值列表。

这个示例通过从 CSV 输入文件中将所有要加载的数据读取和插入到表中，演示了如何规模化地将数据加载到数据库中。这个示例适用于向数据表中添加新行的情况，但是如果想更新表中已有的行，应该怎么做呢？下一个示例就可以解决这个问题。

4.1.2　更新表中记录

前一个示例介绍了使用 CSV 输入文件向数据表中添加新行的方法，因为可以使用循环和 `glob`，所以可以将这个方法扩展到任意数目的文件。但有些时候，不需要向数据表中加载新数据，而是需要更新表中已有的行。

幸运的是，我们可以重新使用从 CSV 输入文件中读取数据的技术来更新表中已有的行。实际上，为 SQL 语句组装一组值和为输入文件中的每一行执行 SQL 语句的技术与前一个示例是一样的。SQL 语句还是有所改变的，不同的是从 `INSERT` 语句变成了 `UPDATE` 语句。

我们已经熟悉了如何使用 CSV 输入文件将数据加载到数据表中，下面学习如何使用 CSV 输入文件更新数据表中已有的记录。要完成这个操作，在文本编辑器中输入下列代码，然后将文件保存为 3db_update_row.py：

```
#!/usr/bin/env python3
import csv
import sqlite3
import sys
# CSV输入文件的路径和文件名
```

```
 6 input_file = sys.argv[1]
 7 # 创建SQLite3内存数据库
 8 # 创建带有4个属性的sales表
 9 con = sqlite3.connect(':memory:')
10 query = """CREATE TABLE IF NOT EXISTS sales
11          (customer VARCHAR(20),
12          product VARCHAR(40),
13          amount FLOAT,
14          date DATE);"""
15 con.execute(query)
16 con.commit()
17 # 向表中插入几行数据
18 data = [('Richard Lucas', 'Notepad', 2.50, '2014-01-02'),
19     ('Jenny Kim', 'Binder', 4.15, '2014-01-15'),
20     ('Svetlana Crow', 'Printer', 155.75, '2014-02-03'),
21     ('Stephen Randolph', 'Computer', 679.40, '2014-02-20')]
22 for tuple in data:
23     print(tuple)
24 statement = "INSERT INTO sales VALUES(?, ?, ?, ?)"
25 con.executemany(statement, data)
26 con.commit()
27 # 读取CSV文件并更新特定的行
28 file_reader = csv.reader(open(input_file, 'r'), delimiter=',')
29 header = next(file_reader, None)
30 for row in file_reader:
31     data = []
32     for column_index in range(len(header)):
33          data.append(row[column_index])
34     print(data)
35     con.execute("UPDATE sales SET amount=?, date=? WHERE customer=?;", data)
36 con.commit()
37 # 查询sales表
38 cursor = con.execute("SELECT * FROM sales")
39 rows = cursor.fetchall()
40 for row in rows:
41     output = []
42     for column_index in range(len(row)):
43          output.append(str(row[column_index]))
44     print(output)
```

所有代码看上去都很熟悉。第 2~4 行代码导入了 3 个 Python 内置模块，这样我们就可以使用其中的方法来读取命令行输入、读取 CSV 输入文件和与内存数据库和数据表进行交互。第 6 行代码将 CSV 输入文件的路径名赋给变量 `input_file`。

第 9~16 行代码创建了一个内存数据库，还创建了一个具有 4 个列属性的数据表 `sales`。

第 18~24 行代码为 `sales` 表创建了 4 条记录，并将这 4 条记录插入到了表中。请用一点时间仔细看一下 Richard Lucas 和 Jenny Kim 的记录。我们稍后要用脚本更新的就是这 2 条记录。现在，`sales` 表中有 4 条记录，看上去与你要更新记录的任何数据表没什么不同，即使比实际的数据表要小很多。

第 28~36 行代码与前面的示例几乎完全相同。唯一的明显区别是第 35 行，`UPDATE` 语句代替了原来的 `INSERT` 语句。在 `UPDATE` 语句中，你必须指定你想更新哪一条记录和哪一个列

属性。在这个例子中，我们想为一组特定的 customer 更新 amount 值和 date 值。像前面的示例一样，UPDATE 语句也需要很多问号占位符表示出查询中的值的位置，CSV 输入文件中数据的顺序也要同查询中属性的顺序一样。在这个例子中，查询中的属性从左到右分别是 amount、date 和 customer；所以，CSV 输入文件中的列从左到右也应该是数量、日期和客户名称。

最后，第 39~44 行代码和前面示例中这部分的代码基本相同。这些代码从 sales 表中取出所有行，然后在命令行窗口或终端窗口中打印出每一行，并使用一个空格分隔每一列。

现在我们需要一个 CSV 输入文件，其中包含着要用来更新数据表中某些记录的数据。下面是创建这个 CSV 文件的步骤：

(1) 打开一个电子表格程序。
(2) 添加如图 4-8 中所示的数据。
(3) 将文件保存为 data_for_updating.csv。

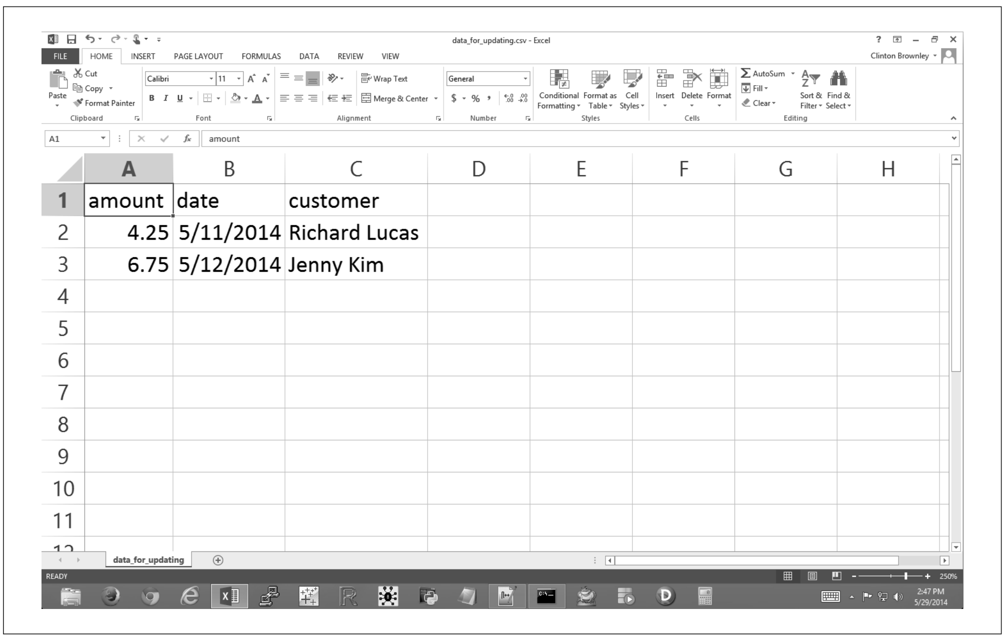


图 4-8：文件 data_for_updating.csv 中的示例数据，显示在 Excel 工作表中

现在你已经完成了 Python 脚本和 CSV 输入文件，可以使用脚本和输入文件来更新 sales 数据表中的某些行了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 3db_update_rows.py data_for_updating.csv
```

图 4-9 展示了将输出打印在命令行窗口中的界面。前 4 行输出（元组）是初始数据行，接下来 2 行（列表）是从 CSV 文件中读入的数据，最后 4 行（列表）是从数据库中取出的更新了行之后的数据。

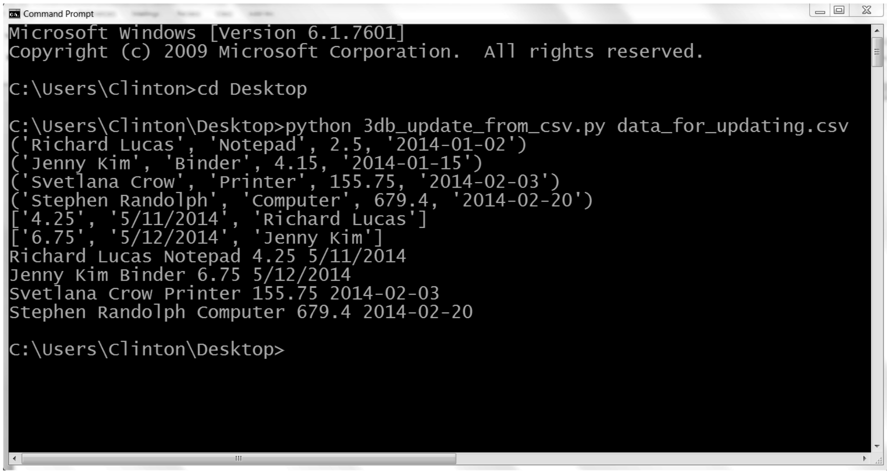


图 4-9：Windows 系统下 3db_update_rows.py 的输出

这个输出首先展示了数据库中初始的 4 行数据，接下来是要更新到数据库中的 2 行数据。在要更新的 2 行数据中，Richard Lucas 的新的 `amount` 值为 `4.25`，新的 `date` 值为 `5/11/2014`。同样，Jenny Kim 的新的 `amount` 值为 `6.75`，新的 `date` 值为 `5/12/2014`。

在这 2 行下面，输出还展示了执行更新之后从数据表中取出的 4 行数据。每行数据打印在一个单独的行中，行中每个数据使用空格分隔。回忆一下，Richard Lucas 的初始 `amount` 值和 `date` 值分别是 `2.5` 和 `2014-01-02`。同样，Jenny Kimd 的初始 `amount` 值和 `data` 值分别是 `4.15` 和 `2014-01-15`。从图 4-9 中的输出可以看出，Richard Lucas 和 Jenny Kim 的这两个值已经被更新成了 CSV 输入文件中的新值。

这个示例演示了批量更新数据表中已有记录的方法，这种方法使用 CSV 输入文件来提供更新特定记录的数据。本章内容到现在为止，提供的示例都是依赖于 Python 内置的 `sqlite3` 模块。使用这个模块可以快速编写脚本，不用依赖某个独立的、需要下载安装的数据库系统，比如 MySQL、PostgreSQL 或 Oracle。在下节中，我们要在下载安装一个数据库系统（MySQL）的基础上重新实现这些示例，还要学习在数据库系统中向数据表中加载数据和更新记录，以及通过数据库查询将输出写入 CSV 文件的方法。下面开始这部分的学习。

4.2　MySQL数据库

要完成本节中的示例代码，需要有 `MySQLdb` 扩展包，也就是 `MySQL-python`（Python v2）或 `mysqlclient`（Python v3）[3]。这个扩展包可以使 Python 与数据库以及数据表进行交互,所以我们使用它与在本节中创建的 MySQL 数据表进行交互。如果你安装了 Anaconda Python，那么你就已经安装了这个扩展包，因为它与安装程序是捆绑在一起的。如果你是从 Python.org 网站上安装的 Python，那么你需要按照附录 A 中的指导步骤安装这个扩展包。

注 3：这些扩展包可以在 Python Package Index（https://pypi.python.org/pypi/mysqlclient）中找到。

和之前一样，为了使用数据库中的表，必须先创建一个。

(1) 参照附录 A，下载安装 MySQL 数据库系统。

下载安装了 MySQL 数据库系统之后，你就可以使用 MySQL 命令行客户端了。

(2) 在命令行中输入 `mysql`，打开 MySQL 命令行客户端。

现在你可以使用命令行界面与 MySQL 数据库系统进行交互了。首先，让我们看一下 MySQL 数据库系统中已有的数据库。

(3) 要完成这个操作，输入以下命令，然后按回车键。图 4-10 展示了 Windows 系统下的结果。

```
SHOW DATABASES;
```

请注意上面的命令以分号结尾，这样 MySQL 才知道你的命令已经输入完成。如果你没有输入分号就按了回车键，那么 MySQL 会期待你继续输入下一行命令（很快你就会看到多行命令）。如果你忘记了分号，也不要着急，在下一行中输入分号，然后按回车键，MySQL 就会执行你的命令。

这条命令的输出显示，在 MySQL 数据库系统中已经有了 4 个数据库。这些数据库使 MySQL 数据库系统能够运行，并包含系统用户的权限信息。要创建一个数据表，必须先创建一个你自己的数据库。

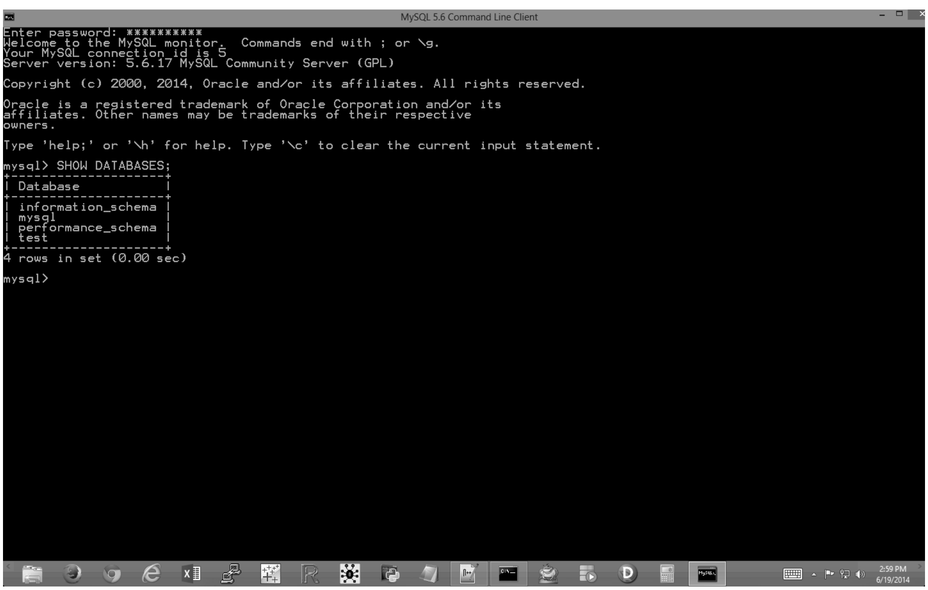


图 4-10：安装了 MySQL 之后，SHOW DATABASES; 命令显示 MySQL 中的默认数据库

(4) 要创建一个数据库，输入以下命令，然后按回车键：

```
CREATE DATABASE my_suppliers;
```

按了回车键之后，你可以再运行一次 `SHOW DATABASE;` 命令，看一下你刚刚创建的新数

据库。要在 my_suppliers 数据库中创建数据表，必须先选择 my_suppliers 数据库。

(5) 要选择 my_suppliers 数据库，输入以下命令，然后按回车键（参见图 4-11）：

```
USE my_suppliers;
```

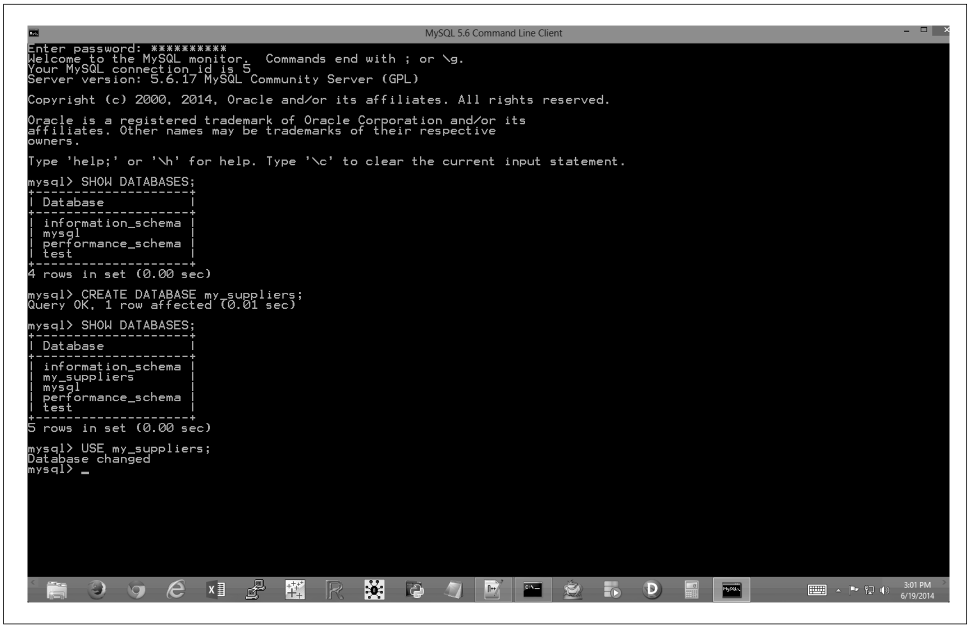


图 4-11：创建一个新数据库 my_suppliers，查看已经包括在已有数据库列表中的新数据库，切换数据库并开始使用

按了回车键之后，你就已经选择了 my_suppliers 数据库。现在可以创建数据表来保存供应商数据了。

(6) 要创建一个数据表 Suppliers，输入以下命令，然后按回车键：

```
CREATE TABLE IF NOT EXISTS Suppliers
(Supplier_Name VARCHAR(20),
Invoice_Number VARCHAR(20),
Part_Number VARCHAR(20),
Cost FLOAT,
Purchase_Date DATE);
```

如果数据库中不存在数据表 Suppliers，这个命令就创建数据表 Suppliers。这个表有 5 个列（也就是 fields 或 attributes）：Supplier_Name、Invoice_Number、Part_Number、Cost 和 Purchase_Date。

前 3 个列是可变字符 VARCHAR 型字段。20 表示为这个字段中的数据分配 20 个字符。如果输入这个字段的数据大于 20 个字符，那么数据将被截断。如果数据少于 20 个字符，那么这个字段就为数据分配一个更小的空间。我们应该为包含变长字符串的字段选择 VARCHAR 类型，因为这样可以使数据表节省保存不必要字符的空间。但是，你必须确定

圆括号中的数值足够大，可以分配足够多的字符以保证字段中最长的字符串不被截断。除了 VARCHAR，还有一些其他字段类型，比如 CHAR、ENUM 和 BLOB。当你想设置一个有固定数量的字符的字段，或者需要将字段中的值向右补齐到一个固定长度时，可以考虑 CHAR 字段类型；当字段取值是一个允许值列表（比如 small、medium、large）时，可以考虑 ENUM 类型字段；当字段内容是长度可变的大量文本时，可以考虑 BLOB 类型的字段。

第四列是一个浮点数 FLOAT 字段。浮点数字段保存浮点数近似值。因为在这个例子中，第四列包含的是货币值，所以可以用 NUMERIC 类型字段替代 FLOAT 类型字段，NUMERIC 字段即定点确定值类型字段。例如，不使用 FLOAT，可以使用 NUMERIC(11, 2)。11 是数值的**精度**，或者是为数值保存的数位总数，包括小数点后面的位数。2 是**小数位数**，也就是小数点后面的数位总数。在这个例子中，我们使用 FLOAT 而不使用 NUMERIC，是为了获得最大的代码可移植性。

最后一列是一个日期 DATE 字段。DATE 字段用来保存日期，形式为 'YYYY-MM-DD'，没有时间部分。所以像 6/19/2014 这样的日期在 MySQL 中被存储为 '2014-06-19'。无效的日期被转换为 '0000-00-00'。

(7) 为了确保数据表创建正确，输入以下命令，然后按回车键：

```
DESCRIBE Suppliers;
```

按了回车键之后，你会看到一个表格，其中列出了你创建的列的名称、每列的数据类型（例如：VARCHAR 或 FLOAT）和列中的值是否可以为 NULL。

现在我们已经创建了数据库 my_suppliers 和数据表 Suppliers，还要创建一个新用户，并授予这个用户与数据库和数据表进行交互的权限。

(8) 要创建一个新用户，输入以下命令，然后按回车键（请注意用你要使用的用户名替换 *username*；你还应该用自己的密码替换 *secret_password*，来获得更高的安全性）：

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'secret_password';
```

我们已经创建了一个新用户，现在要为这个用户授予 my_suppliers 数据库的所有权限。通过授予用户所有的数据库权限，就使这个用户可以在数据库中的表上执行各种操作。这些权限非常有用，因为本节中脚本涉及的操作包括向表中加载数据、修改表中特定记录和对数据表执行查询。

(9) 要向新用户授予所有权限，输入以下两条命令，然后在每条命令后面按回车键（同样，注意用你在前一步创建的用户名替换 *username*）：

```
GRANT ALL PRIVILEGES ON my_suppliers.* TO 'username'@'localhost';
FLUSH PRIVILEGES;
```

现在你可以同本地主机（也就是你自己的计算机）中的 my_suppliers 数据库中的 Suppliers 表进行交互了。参见图 4-12。



图 4-12：在 my_suppliers 数据库中创建新表 Suppliers，创建一个新用户，为新用户授予 my_suppliers 数据库和其中的表的所有权限

现在你已经创建了一个数据库和用来保存数据的表，下面就开始学习如何使用 Python 向表中加载数据。

4.2.1　向表中插入新记录

现在我们准备从一个 CSV 文件中将记录加载到数据表中。你已经可以从 Python 脚本或 Excel 文件中将记录输出到 CSV 文件，这可以使你创建一个多用途的数据通道。

下面创建一个新的 Python 脚本。这个脚本会将数据从 CSV 文件中插入到我们的数据表，然后展示表中的数据。其中的第二个步骤，即在命令行窗口或终端窗口打印数据，并不是必须的（而且如果你加载了上千条记录的话，我也不建议将记录打印在窗口中），我之所以在示例中包括了这个步骤，是想演示一种不指定单独的列索引，打印出每条记录中所有列的方法（也就是说，这种语法可以扩展到任意数目的列）。

首先，在文本编辑器中输入下列代码，然后将文件保存为 4db_mysql_load_from_csv.py：

```
1 #!/usr/bin/env python3
2 import csv
3 import MySQLdb
4 import sys
5 from datetime import datetime, date
6 
7 # CSV输入文件的路径和文件名
8 input_file = sys.argv[1]
9 # 连接MySQL数据库
10 con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
11 user='root', passwd='my_password')
12 c = con.cursor()
13 # 向Suppliers表中插入数据
14 file_reader = csv.reader(open(input_file, 'r', newline=''))
15 header = next(file_reader)
16 for row in file_reader:
17     data = []
18     for column_index in range(len(header)):
19         if column_index < 4:
20             data.append(str(row[column_index]).lstrip('$')\
21             .replace(',', '').strip())
22         else:
23             a_date = datetime.date(datetime.strptime(\
24             str(row[column_index]), '%m/%d/%Y'))
25             # %Y: year is 2015; %y: year is 15
26             a_date = a_date.strftime('%Y-%m-%d')
27             data.append(a_date)
28     print data
29     c.execute("""INSERT INTO Suppliers VALUES (%s, %s, %s, %s, %s);""", data)
30 con.commit()
31 print("")
32 # 查询Suppliers表
33 c.execute("SELECT * FROM Suppliers")
34 rows = c.fetchall()
35 for row in rows:
36     row_list_output =[]
37     for column_index in range(len(row)):
38         row_list_output.append(str(row[column_index]))
39     print(row_list_output)
```

这个脚本和第 2 章中的脚本一样，依赖于 csv、datetime、string 和 sys 模块。第 2 行代码导入 csv 模块，这样我们就可以使用其中的方法读取和解析 CSV 文件了。第 4 行代码导入 sys 模块，以使我们可以在命令行中提供一个文件的路径和名称供脚本使用。第 5 行代码从 datetime 模块导入 datetime 和 date 方法，以使我们可以对输入文件中最后一列的日

期数据进行处理和格式化。这里需要剥离数据中的美元符号，还要删除任何内嵌的逗号，这样数据才能被输入到数据表中接受浮点数的字段中。第 3 行代码导入之前下载并安装好的 MySQLdb 扩展模块，以使我们可以使用其中的方法来连接 MySQL 数据库和数据表。

第 8 行代码使用 sys 模块在命令行中读取文件的路径和名称，并将这个值赋给变量 input_file。

第 10 行代码使用 MySQLdb 模块的 connect() 方法连接 my_suppliers，即在上一节中创建的 MySQL 数据库。与处理 CSV 和 Excel 文件不同（对这两种文件，你是对文件本身进行读取、修改或删除操作），MySQL 建立的数据库就像一台独立计算机（**服务器**），你可以向数据库请求连接、发送数据和请求数据。在连接时，需要指定一些通用参数，包括 host、port、db、user 和 passwd。

host 是数据库所在的机器的主机名。在本例中，MySQL 服务器保存在你的计算机上，所以 host 是 localhost。当你连接其他数据源时，服务器可能位于不同的机器上，所以你需要修改 localhost，更换成服务器所在的机器的主机名。

port 是 MySQL 服务器的 TCP/IP 连接端口号。我们要使用的端口号是默认的端口号 3306。和 host 参数一样，如果你不在本地主机上工作，而且你的 MySQL 服务器管理员为服务器设置了不同的端口号，那么你必须使用这个端口号去连接 MySQL 服务器。本例中使用了值安装 MySQL 服务器，所以 localhost 是有效的主机名，3306 是有效的端口号。

db 是你想连接的数据库名称。在本例中，我们想连接 my_suppliers 数据库，因为它保存着我们要加载数据的表。如果以后你在本地计算机上创建了另一个数据库，比如 contacts，那么就必须将 db 参数从 my_suppliers 修改为 contacts，来连接这个数据库。

user 是进行数据库连接的用户的用户名。在本例中，我们作为“root”用户进行连接，使用的密码就是在安装 MySQL 服务器时创建的密码。当你安装 MySQL 时（你可以按照附录 A 中的指示步骤完成安装），MySQL 安装程序会要求你为根用户提供密码。在本例中，我们为根用户创建的密码，也就是在代码中提供给 passwd 参数的密码，即 'my_password'。当然，如果你在安装 MySQL 时为根用户创建了不同的密码，那么你应该在脚本代码中使用你自己的密码替换掉 'my_password'。

在创建数据库、表和新用户的步骤中，我创建了一个新用户 clinton，密码为 secret_password。因此，我可以在下列脚本：user='clinton' and passwd='secret_password' 中使用这个连接。如果你想在代码中保留 user='root'，那么你应该用在安装 MySQL 服务器时实际设置的密码替换掉 'my_password'。或者，如果你使用 CREATE USER 命令创建了新用户，也可以使用这个新用户的用户名和密码。通过这 5 个参数，你可以创建与 my_suppliers 数据库的本地连接。

第 12 行代码创建了一个光标，我们可以使用它来在 my_suppliers 数据库中对 Suppliers 表执行 SQL 语句，并将修改提交到数据库。

第 14~29 行代码从 CSV 文件中读取要加载到数据表中的数据，并对输入文件中的每行数据执行一条 SQL 语句，将其插入数据表中。第 14 行代码使用 csv 模块创建了 file_reader 对象。第 15 行代码使用 next() 函数从输入文件中读出第一行，也就是标题行，并将其赋

给变量 header。第 16 行代码创建了一个 for 循环，在输入文件的所有数据行之间循环。第 17 行代码创建了一个空列表变量 data，对于输入文件中的每一行，用行中的数据去填充 data，这些数据将在第 29 行代码中的 INSERT 语句中使用。第 18 行代码创建了一个 for 循环，在每行数据中的各个列之间循环。第 19 行代码创建了一个 if-else 语句，检验列索引是否小于 4。因为输入文件中有 5 列，而且日期在最后一列，所以日期列的索引值为 4。因此，这行代码判断我们是否在处理日期列前面的列。对于所有日期列前面的列，索引值为 0、1、2 和 3，第 20 行代码将列中的值转换为字符串，如果字符串左侧有美元符号，就剥离掉这个字符，然后将这个值追加到列表变量 data 中。对于最后的日期列，第 23 行代码将其转换为字符串，并使用字符串按照代码中的格式创建一个 datetime 对象，然后将 datetime 对象转换成一个 data 对象（只保留年、月、日），最后将这个值赋给变量 a_date。此后，第 26 行代码将这个 data 对象转换成一个字符串，使用的新格式是要加载到 MySQL 数据库中的日期字符串格式（就是 YYYY-MM-DD），并将格式化好的字符串重新赋给变量 a_date。最后，第 27 行代码将这个字符串追加到 data 中。

第 28 行代码将追加到 data 中的数据打印到命令行窗口或终端窗口上。请注意缩进。这行代码的缩进是在外部 for 循环之下，不是在内部 for 循环之下的，所以它是对于输入文件中的每一行来执行，而不是对于输入文件中的每一行和每一列去执行。这一行有助于调试脚本，但是一旦你确定代码能够正确运行，就可以将这行代码删除或者注释掉，这样在屏幕上就不会打印出过多的输出了。

第 29 行代码是实际将每行数据加载到数据表中的代码。这行代码使用光标对象的 execute() 方法执行一条 INSERT 语句，将一行数据插入到表 Suppliers 中。每个 %s 都是要插入的实际值的占位符。占位符的数量对应着输入文件中列的数量，它们都对应着数据表中列的数量。而且，输入文件中列的顺序也要对应数据表中列的顺序。要替换到 %s 位置的值来自于列表 data，这个列表要放在 execute() 语句中逗号的后面。因为 data 是使用输入文件中的每行数据填充的，而且 INSERT 语句也是对于输入文件中的每行数据执行的，所以这些代码可以高效地从输入文件中读取数据行，并把这些数据行加载到数据表中。再强调一次，要注意缩进。这行代码是在外部 for 循环之下缩进的，所以它对于输入文件中的每一行数据执行一次。最后，第 30 行代码是另一个 commit 语句，将修改提交到数据库。

第 33~39 行代码演示了如何从数据表 Suppliers 中选择所有数据，并将输出打印到命令行窗口或者终端窗口。第 33 和 34 行代码执行一条 SQL 语句，从 Suppliers 表中选择所有数据，并将输出中的所有行读入变量 rows。第 35 行代码创建了一个 for 循环，在 rows 的每一行中循环。第 36 行代码创建了一个空列表变量 row_list_output，用来保存 SQL 查询输出中每一行所有的值。第 37 行代码创建了一个 for 循环，在每行的各个列之间循环。第 38 行代码将每个值都转换成一个字符串，然后追加到 row_list_output 中。最后，当行中所有的值都进入 row_list_output 中后，第 39 行代码将这一行打印到屏幕上。

现在我们已经完成了 Python 脚本，可以使用这个脚本将 supplier_data.csv 中的数据加载到 Suppliers 数据表中了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 4db_mysql_load_from_csv.py supplier_data.csv
```

在 Windows 系统中，你可以看到如图 4-13 所示的输出被打印到命令行窗口中。输出中的第一部分是从 CSV 文件中解析出的数据，第二部分是从数据表中查询出的同样的数据。

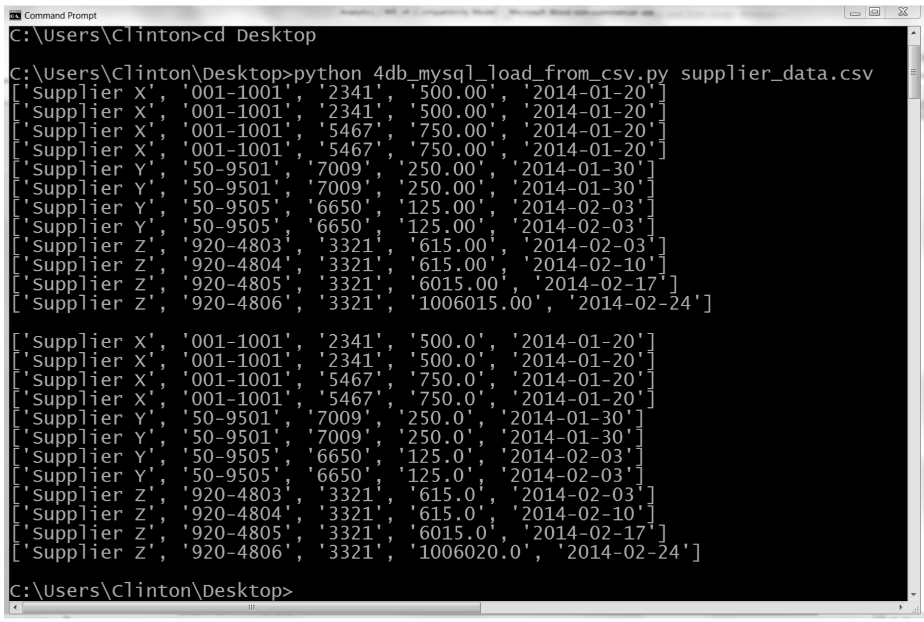


图 4-13：输出显示 supplier_data.csv 中的数据就是插入到 MySQL 数据表 Suppliers 中的数据

这个输出展示了 12 个值列表，来自于 CSV 输入文件中除标题行之外的 12 行数据。你可以识别出这 12 个列表，因为每个列表都在方括号（[]）之间，列表中的每个值以逗号分隔。

在从 CSV 文件中读取的 12 个输入数据列表下面，有一个空行，然后是从数据表中取出的 12 行输出数据，使用的查询语句为 `SELECT * FROM Suppliers`。每行数据占一行，行中的值以逗号分隔。这个输出证明了数据被成功地加载到了 `Suppliers` 表中，并被成功读出。

如果要以其他方式确定结果，可以在 MySQL 命令行客户端中输入以下命令，然后按回车键：

```
SELECT * FROM Suppliers;
```

按了回车键之后，你会看到一个表格，其中列出了 `Suppliers` 数据表中所有的列，以及每列中的 12 行数据，如图 4-14 所示。

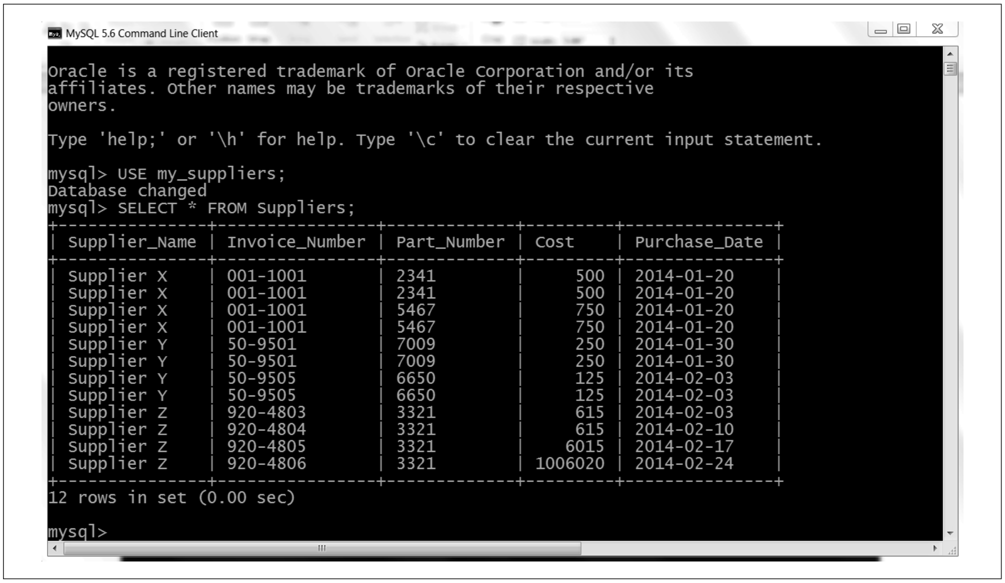


图 4-14：使用 MySQL 命令行客户端在 Suppliers 表中进行数据查询的结果

现在我们已经有了一个充满数据的数据表，下面就开始学习如何在数据库中进行查询，并使用 Python 将查询结果写入 CSV 文件，而不是打印在屏幕上。

4.2.2　查询一个表并将输出写入CSV文件

数据表中有了数据之后，最常见的下一个步骤就是使用查询从表中取出一组数据，用来进行分析或满足某种商业需求。例如，你可能想知道哪些客户提供了最多的利润，或者哪些费用超过了具体的阈值。

下面创建一个新的 Python 脚本。这个脚本会从 `Suppliers` 数据表中查询出一组特定记录，然后将输出写入 CSV 输出文件。在这个例子中，我们想找出 Cost 列中的值大于 1000.00 的所有记录，并将这些记录所有列中的值输出。首先，在文本编辑器中输入下列代码，然后将文件保存为 5db_mysql_write_to_file.py：

```
#!/usr/bin/env python3
import csv
import MySQLdb
import sys
# CSV输出文件的路径和文件名
output_file = sys.argv[1]
# 连接MySQL数据库
con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
user='root', passwd='my_password')
c = con.cursor()
# 创建写文件的对象,并写入标题行
filewriter = csv.writer(open(output_file, 'w', newline=''), delimiter=',')
header = ['Supplier Name','Invoice Number','Part Number','Cost','Purchase Date']
```

```
14 filewriter.writerow(header)
15 # 查询Suppliers表,并将结果写入CSV输出文件
16 c.execute("""SELECT *
17         FROM Suppliers
18         WHERE Cost > 700.0;""")
19 rows = c.fetchall()
20 for row in rows:
21     filewriter.writerow(row)
```

这个示例中的代码几乎就是前一个示例中代码的子集，所以下面将重点介绍其中的新代码。

第 2、3 和 4 行代码分别导入 csv、MySQLdb 和 sys 模块，这样我们就可以使用其中的方法来与 MySQL 数据库进行交互，并将查询结果写入一个 CSV 文件了。

第 6 行代码使用 sys 模块在命令行中读取文件的路径和名称，并将其赋给变量 output_file。

第 8 行代码使用 MySQLdb 的 connect() 方法连接到 my_suppliers，就是我们在本章前面创建的 MySQL 数据库。第 10 行代码创建了一个光标，用来在 my_suppliers 数据库中的 Suppliers 数据表上执行 SQL 语句，并将修改提交到数据库。

第 12 行代码使用 csv 模块的 wirter() 方法创建了 filewriter 对象。

第 13 行代码创建了一个列表变量 header，其中包含 5 个字符串，对应着数据表中的列标题。第 14 行代码使用 filewriter 的 writerow() 方法将这个由逗号分隔的字符串列表写入 CSV 格式的输出文件。数据库查询只输出数据，不输出列标题，所以这些代码可以确保输出文件中的各列有列标题。

第 16~18 行代码是数据库查询，选择 Cost 列中的值大于 700.0 的那些行，并选择这些行中所有的列。因为包含在 3 个双引号之间，所以这个查询可以分布在多行之内。用 3 个双引号封装查询非常有助于将查询写成易读的形式。

第 19~21 行代码与前一个示例中的代码非常相似，不同之处是前一个示例将输出打印到命令行窗口或终端窗口，这个示例的第 21 行代码将输出写入 CSV 格式的输出文件。

我们完成了 Python 脚本，现在可以使用脚本从 Suppliers 数据表中查询出特定数据，并将查询结果写入 CSV 格式的输出文件。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 5db_mysql_write_to_file.py output_files\5output.csv
```

在命令行窗口或终端窗口中你不会看到任何输出，但是你可以打开输出文件 5output.csv 查看一下结果。

你可以看到，输出文件中包含一个标题行，其中有 5 个列标题，文件中还有数据表中的 4 行数据，其中 Cost 列中的值大于 700.0。Excel 会将购买日期列中的日期格式化为 MM/DD/YYYY，Cost 列中的值不包含逗号或美元符号，如果需要的话，对这些值进行格式化也非常容易。

向数据表中加载数据和从数据表中查询数据是对数据表常用的两种操作。另外一种常用操

作是更新数据表中已有的行。下一个示例将处理这种情况，介绍如何更新表中已有的行。

4.2.3　更新表中记录

上一个示例介绍了如何使用 CSV 输入文件向一个 MySQL 数据表中批量添加数据，以及如何将 SQL 查询结果写入 CSV 输出文件。但有些时候，我们不需要向表中加载新数据或进行查询，而是需要更新表中已有的行。

幸运的是，我们可以重新使用前面介绍的技术，从 CSV 输入文件中读取数据来更新表中的行。实际上，我们要为 SQL 语句组合一行数据，然后对于 CSV 输入文件中的每一行数据运行一次 SQL 语句，这种技术与前一个示例是完全一样的。只是 SQL 语句有所变化，从 `INSERT` 语句变为了 `UPDATE` 语句。

我们已经熟悉了如何使用 CSV 输入文件将数据加载到数据表中，下面学习如何使用 CSV 输入文件更新 MySQL 数据表中已有的记录。要完成这个操作，在文本编辑器中输入下列代码，然后将文件保存为 6db_mysql_update_from_csv.py：

```
 1 #!/usr/bin/env python3
 2 import csv
 3 import MySQLdb
 4 import sys
 5
 6 # CSV输入文件的路径和文件名
 7 input_file = sys.argv[1]
 8 # 连接MySQL数据库
 9 con = MySQLdb.connect(host='localhost', port=3306, db='my_suppliers', \
10 user='root', passwd='my_password')
11 c = con.cursor()
12
13 # 读取CSV文件并更新特定的行
14 file_reader = csv.reader(open(input_file, 'r', newline=''), delimiter=',')
15 header = next(file_reader, None)
16 for row in file_reader:
17     data = []
18     for column_index in range(len(header)):
19          data.append(str(row[column_index]).strip())
20     print(data)
21     c.execute("""UPDATE Suppliers SET Cost=%s, Purchase_Date=%s \
22     WHERE Supplier_Name=%s;""", data)
23 con.commit()
24 # 查询Suppliers表
25 c.execute("SELECT * FROM Suppliers")
26 rows = c.fetchall()
27 for row in rows:
28     output = []
29     for column_index in range(len(row)):
30          output.append(str(row[column_index]))
31     print(output)
```

这个示例中的所有代码看上去都很熟悉。第 2~4 行代码导入了 3 个 Python 内置模块，这样我们就可以使用其中的方法来读取 CSV 输入文件、与 MySQL 数据库进行交互和读取命令

行输入。第 7 行代码将 CSV 输入文件的路径名赋给变量 `input_file`。

第 9 行代码与 `my_suppliers` 数据库建立连接，使用与前一个示例中同样的连接参数。第 11 行代码创建了一个光标对象，用来执行 SQL 查询，并将修改提交到数据库。

第 15~24 行代码与本章第一个示例中的代码几乎是相同的。唯一的明显区别是在第 21 行，`UPDATE` 语句代替了原来的 `INSERT` 语句。在 `UPDATE` 语句中，你必须指定你想更新哪一条记录和哪一个列属性。在这个例子中，我们想为一组特定的 Supplier Names 更新 Cost 值和 Purchase Date 值。像前面的示例一样，`UPDATE` 语句中需要几个值，就需要几个 `%s` 占位符表示出查询中的值的位置，CSV 输入文件中数据的顺序也要同查询中属性的顺序一样。在这个例子中，查询中的属性从左到右分别是 `Cost`、`Purchase_Date` 和 `Supplier_Name`；所以，CSV 输入文件中的列从左到右也应该是成本、购买日期和供应商姓名。

最后，第 25~31 行代码和前面示例中这部分的代码基本相同。这些代码从 `Suppliers` 表中取出所有行，然后在命令行窗口或终端窗口中打印出每一行，并使用一个空格分隔每一列。

现在我们需要一个 CSV 输入文件，其中包含着要用来更新数据表中某些记录的数据：

(1) 打开 Excel。
(2) 添加如图 4-15 中所示的数据。
(3) 将文件保存为 data_for_updating_mysql.csv。

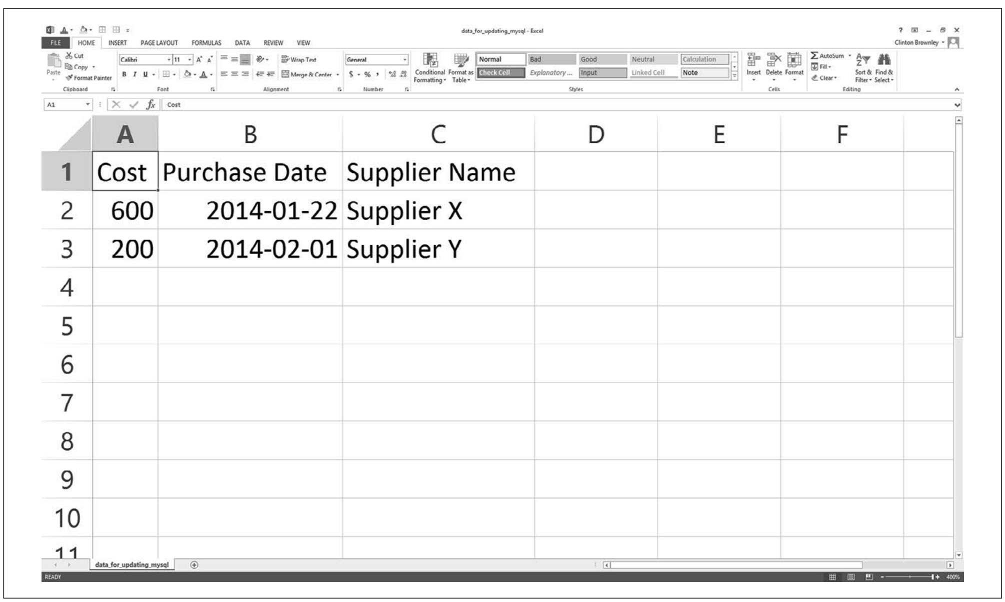


图 4-15：文件 data_for_updating_mysql.csv 中的示例数据，显示在 Excel 工作表中

现在你已经完成了 Python 脚本和 CSV 输入文件，可以使用脚本和输入文件来更新 `Suppliers` 数据表中的某些行了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 6db_mysql_update_from_csv.py data_for_updating_mysql.csv
```

在 Windows 系统中，你可以看到如图 4-16 所示的输出被打印到命令行窗口上。前两行是 CSV 文件中的数据，其余各行是记录更新之后数据表中的数据。

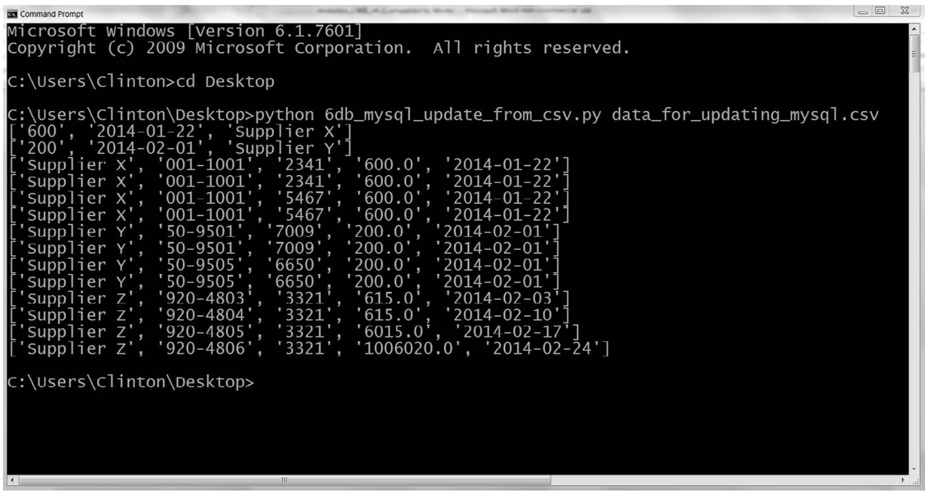


图 4-16：使用 CSV 文件中的数据更新 MySQL 数据表中的行

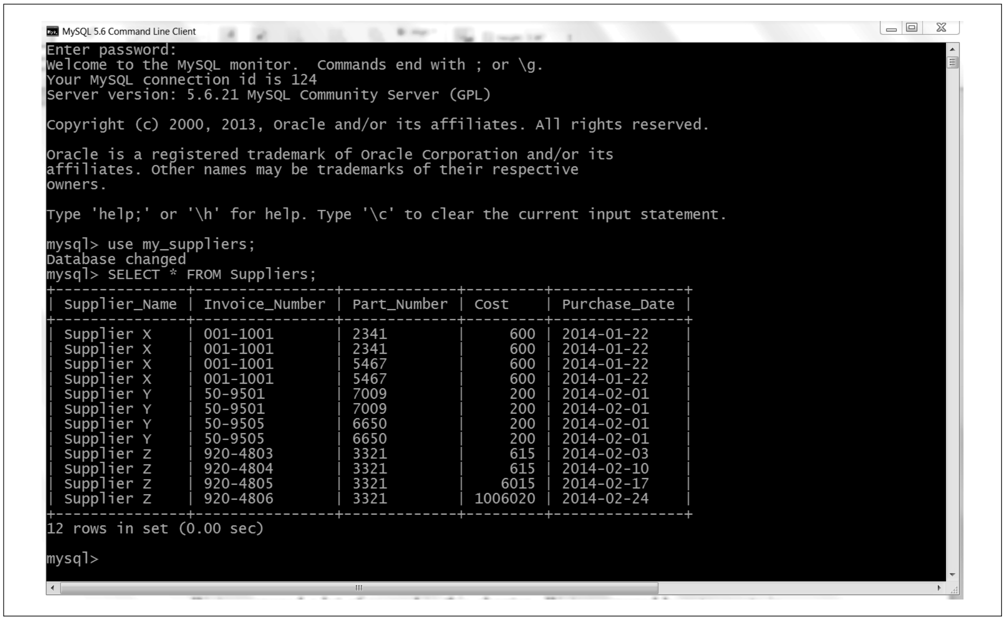
这个输出展示了来自于 CSV 输入文件中除标题行之外的两行数据。你可以识别出这两个列表，因为每个列表都包含在方括号（[]）之间，列表中的每个值以逗号分隔。对于 Supplier X，Cost 值为 600，Purchase Date 值为 2014-01-22。对于 Supplier Y，Cost 值为 200，Purchase Date 值为 2014-02-01。

在这两行下面，输出还展示了执行更新之后从数据表中取出的 12 行数据。每行数据占一行，行中的值由空格分隔。回忆一下，Supplier X 原来的 Cost 值和 Purchase Date 值分别是 500、750 和 2014-01-20。同样，Supplier Y 原来的 Cost 值和 Purchase Date 值分别是 250、125 和 2014-01-30、2013-02-03。从打印在命令行窗口中的输出可以看出，Supplier X 和 Supplier Y 中的这些值已经被修改成了 CSV 输入文件中提供的新值。

为了确认 MySQL 数据表中与 Supplier X 和 Supplier Y 相关的 8 行数据已经被更新，现在回到 MySQL 命令行客户端，输入以下命令，然后按回车键：

```
SELECT * FROM Suppliers;
```

按了回车键之后，你可以看到一个表格，其中列出了 `Suppliers` 数据表中的各列，以及每列中的 12 行数据，如图 4-17 所示。你会看到与 Supplier X 和 Supplier Y 相关的 8 行记录已经被更新为 CSV 输入文件中提供的数据。



```
MySQL 5.6 Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 124
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use my_suppliers;
Database changed
mysql> SELECT * FROM Suppliers;
+---------------+----------------+-------------+---------+---------------+
| Supplier_Name | Invoice_Number | Part_Number | Cost    | Purchase_Date |
+---------------+----------------+-------------+---------+---------------+
| Supplier X    | 001-1001       | 2341        |     600 | 2014-01-22    |
| Supplier X    | 001-1001       | 2341        |     600 | 2014-01-22    |
| Supplier X    | 001-1001       | 5467        |     600 | 2014-01-22    |
| Supplier X    | 001-1001       | 5467        |     600 | 2014-01-22    |
| Supplier Y    | 50-9501        | 7009        |     200 | 2014-02-01    |
| Supplier Y    | 50-9501        | 7009        |     200 | 2014-02-01    |
| Supplier Y    | 50-9505        | 6650        |     200 | 2014-02-01    |
| Supplier Y    | 50-9505        | 6650        |     200 | 2014-02-01    |
| Supplier Z    | 920-4803       | 3321        |     615 | 2014-02-03    |
| Supplier Z    | 920-4804       | 3321        |     615 | 2014-02-10    |
| Supplier Z    | 920-4805       | 3321        |    6015 | 2014-02-17    |
| Supplier Z    | 920-4806       | 3321        | 1006020 | 2014-02-24    |
+---------------+----------------+-------------+---------+---------------+
12 rows in set (0.00 sec)

mysql>
```

图 4-17：Suppliers 表中的记录更新后，使用 MySQL 命令行客户端进行查询的结果

本章介绍了很多基础知识，不仅包括如何使用 `sqlite3` 创建内存数据库和持久化数据库，以及如何与数据库中的数据表进行交互，还包括创建 MySQL 数据库和数据表、使用 Python 访问 MySQL 数据库和数据表、从 CSV 文件中向 MySQL 数据表中加载数据、使用 CSV 文件中的数据更新 MySQL 数据表中的记录，以及将查询结果写入 CSV 输出文件的方法。如果你一直跟随本章内容练习示例代码，那么你应该完成了 6 个新的 Python 脚本！

练习本章中示例代码的最大收获是，你现在已经很好地掌握了存取数据库中数据的技术，而数据库是商业中最常用的一种数据存储工具。本章重点介绍了 MySQL 数据库系统，但是我们在开头曾提到过，现在有很多其他数据库系统也用于商业中。例如，你可以了解一下 PostgreSQL 数据库系统（http://www.postgresql.org），在 Psycopg（http://initd.org/psycopg）和 PyPI（https://pypi.python.org/pypi/psycopg2）这两个网站，你都可以找到关于 PostgreSQL 的通用 Python 连接适配器的信息。同样，你也可以了解一下 Oracle 数据库系统（https://www.oracle.com/database/index.html），在 SourceForge（http://cx-oracle.sourceforge.net）和 PyPI（https://pypi.python.org/pypi/cx_Oracle）这两个网站，你可以找到关于 Oracle 连接适配器的信息。此外，还有一个常用的 Python SQL 工具箱，名为 SQLAlchemy（http://www.sqlalchemy.org），可以同时支持 Python2 和 Python3，其中包括 SQLite、MySQL、PostgreSQL、Oracle 和若干其他数据库系统的适配器。

到此为止，我们学习了在 CSV 文件、Excel 工作簿和数据库中存取、浏览和处理数据的方法，这是商业中最常用的 3 种数据源。下一步，我们将介绍几个应用程序，看看如何综合运用这些技术解决具体问题。首先，我们讨论如何在一个大的文件集合中找到一组特定的项目。其次，第二个应用程序演示了在输入文件中为任意数目的分类计算统计量的方法。

最后，第三个应用程序演示了如何分析文本文件并为任意数目的分类计算统计量。在学习了这些示例之后，你应该可以掌握如何综合运用在本书中学到的技能来解决具体问题了。

4.3　本章练习

(1) 练习从 CSV 文件向数据表中加载数据，创建一个新表和一个新的输入文件，并编写一个新的 Python 脚本将输入数据加载到表中，使用 SQLite3 数据库和 MySQL 数据库都可以。

(2) 练习在 SQLite3 数据库或 MySQL 数据库中对数据表执行查询，并将查询结果写入 CSV 输出文件。创建一个新的 Python 脚本，其中包含一个新的数据库查询，从你创建的一个数据表中提取数据。参考并修改 MySQL 那一节中演示如何写入输出文件的脚本，将标题行和你查询出的数据写入输出文件。

(3) 练习使用 CSV 文件中的数据更新数据表中的记录，使用 SQLite3 数据库和 MySQL 数据库都可以。创建一个新表，并向表中加载数据。创建一个新的 CSV 文件，保存用来更新表中特定记录的数据。创建一个新的 Python 脚本，使用 CSV 文件中的数据更新表中特定的记录。

第 5 章

应用程序

5.1 在一个大文件集合中查找一组项目

每个公司都在各自不同的业务过程中积累了大量文件，在供应商、客户、内部运营和其他业务过程中，可能存在着大量历史文件。正如本书已经讨论过的，这些数据可能被保存在像 CSV 文件这样的带分隔符的平面文件中，可能被保存在 Excel 工作簿和电子表格中，也可能被保存在其他存储系统中。保存这些文件是有价值的，因为它们可以提供数据用来分析，有助于跟踪随着时间发生的变化，还可以为决策提供支持。

但是当你有大量历史数据的时候，要找到真正需要的数据是非常困难的。想象一下，假如你有 300 个 Excel 工作簿和 200 个 CSV 文件（你一直交替使用这两种类型的文件），里面包含着过去 5 年有过购买行为的供应商数据。现在你和一个供应商正在进行谈判，你想找到一些对这次谈判有用的信息的历史记录。

当然，你可以打开每个文件，找出你需要的记录，然后将记录复制粘贴到一个新文件中。但是，这个过程太痛苦了，既浪费时间，又容易出错。这正是展示你刚学会的 Python 编程技能的一个绝好机会，你可以自动化地完成整个过程，既节省时间，又不会出错。

为了模拟在一个包含了几百个 Excel 工作簿和 CSV 文件的历史文件夹中进行搜索，我们需要创建这个历史文件夹，以及一些 Excel 工作簿和 CSV 文件。要完成这个操作，需要按以下步骤进行。

(1) 切换到桌面。
(2) 在桌面上点鼠标右键。
(3) 选择“新建”，然后选择“文件夹”，在桌面上新建一个文件夹。
(4) 将新建的文件夹命名为“file_archive”。

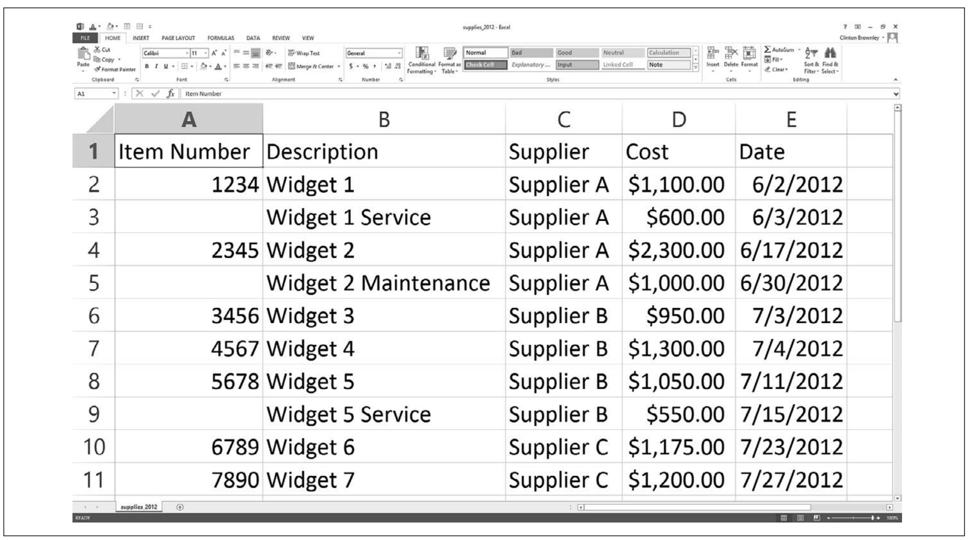
现在你的桌面上应该有一个名为 file_archive 的新文件夹，如图 5-1 所示。



图 5-1：在桌面上新建文件夹 file_archive

(5) 打开 Excel，输入图 5-2 中的数据。

这个 CSV 文件有 5 列：Item Number、Description、Supplier、Cost 和 Date。你可以看到在第一列中是只有部件才有项目编号。部件服务和部件维修有独立的记录，但是服务记录和维修记录中没有项目编号。



	A	B	C	D	E
1	Item Number	Description	Supplier	Cost	Date
2	1234	Widget 1	Supplier A	$1,100.00	6/2/2012
3		Widget 1 Service	Supplier A	$600.00	6/3/2012
4	2345	Widget 2	Supplier A	$2,300.00	6/17/2012
5		Widget 2 Maintenance	Supplier A	$1,000.00	6/30/2012
6	3456	Widget 3	Supplier B	$950.00	7/3/2012
7	4567	Widget 4	Supplier B	$1,300.00	7/4/2012
8	5678	Widget 5	Supplier B	$1,050.00	7/11/2012
9		Widget 5 Service	Supplier B	$550.00	7/15/2012
10	6789	Widget 6	Supplier C	$1,175.00	7/23/2012
11	7890	Widget 7	Supplier C	$1,200.00	7/27/2012

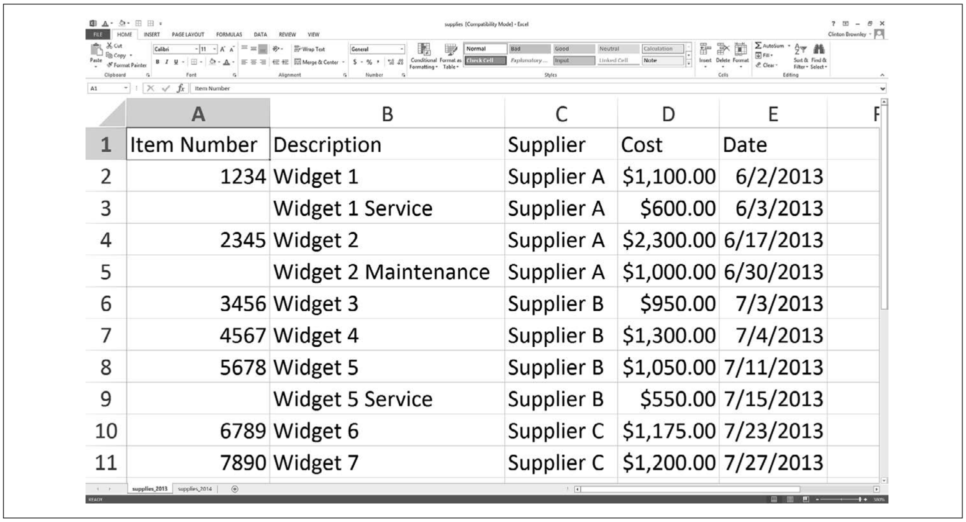
图 5-2：CSV 文件 supplies_2012.csv 中的示例数据，显示在 Excel 工作表中

(6) 将文件保存在 file_archive 文件夹中，命名为 supplies_2012.csv。

好了，现在我们有了一个 CSV 文件。下一步，需要创建一个 Excel 工作簿。要快速完成这个操作，可以使用刚建好的 CSV 文件。

(7) 在工作表 supplies_2012 中，将 Date 列中的日期由 2012 改为 2013。

工作表现在应该如图 5-3 所示。你可以看到，只有日期发生了变化。



	A	B	C	D	E
1	Item Number	Description	Supplier	Cost	Date
2	1234	Widget 1	Supplier A	$1,100.00	6/2/2013
3		Widget 1 Service	Supplier A	$600.00	6/3/2013
4	2345	Widget 2	Supplier A	$2,300.00	6/17/2013
5		Widget 2 Maintenance	Supplier A	$1,000.00	6/30/2013
6	3456	Widget 3	Supplier B	$950.00	7/3/2013
7	4567	Widget 4	Supplier B	$1,300.00	7/4/2013
8	5678	Widget 5	Supplier B	$1,050.00	7/11/2013
9		Widget 5 Service	Supplier B	$550.00	7/15/2013
10	6789	Widget 6	Supplier C	$1,175.00	7/23/2013
11	7890	Widget 7	Supplier C	$1,200.00	7/27/2013

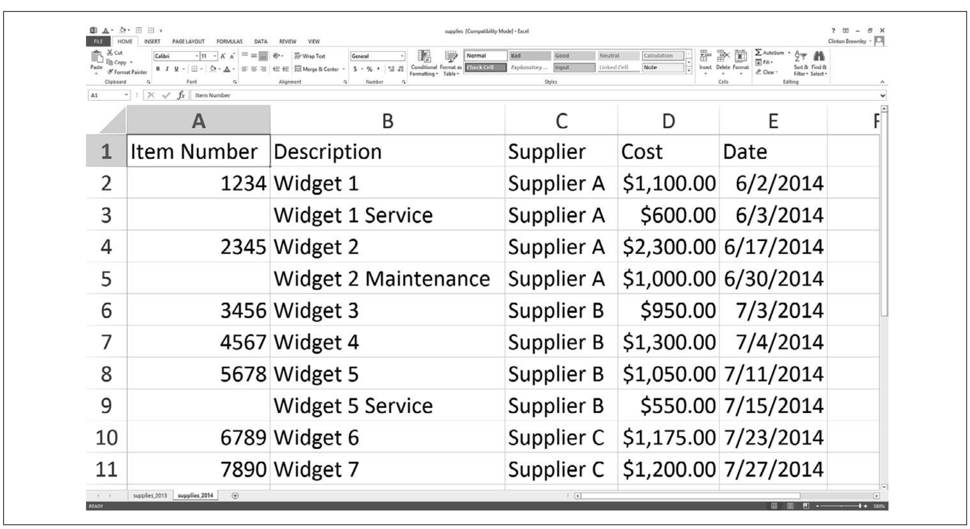
图 5-3：将 supplies_2012 中的日期由 2012 改成 2013，添加一个 2013 年的工作表

(8) 将工作表的名称改为 supplies_2013。

为了使这个文件成为一个具有多个工作表的工作簿，下面再添加一个新的工作表。

(9) 点击左下角的 + 按钮，添加一个新的工作表。
(10) 将新工作表命名为 supplies_2014。
(11) 将 supplies_2013 工作表中的数据复制粘贴到 supplies_2014 工作表中。
(12) 将 Date 列中的日期由 2013 修改为 2014。

supplies_2014 工作表如图 5-4 所示。



	A	B	C	D	E
1	Item Number	Description	Supplier	Cost	Date
2	1234	Widget 1	Supplier A	$1,100.00	6/2/2014
3		Widget 1 Service	Supplier A	$600.00	6/3/2014
4	2345	Widget 2	Supplier A	$2,300.00	6/17/2014
5		Widget 2 Maintenance	Supplier A	$1,000.00	6/30/2014
6	3456	Widget 3	Supplier B	$950.00	7/3/2014
7	4567	Widget 4	Supplier B	$1,300.00	7/4/2014
8	5678	Widget 5	Supplier B	$1,050.00	7/11/2014
9		Widget 5 Service	Supplier B	$550.00	7/15/2014
10	6789	Widget 6	Supplier C	$1,175.00	7/23/2014
11	7890	Widget 7	Supplier C	$1,200.00	7/27/2014

图 5-4：supplies_2014 工作表

你可以看到，只有日期发生了变化，也就是说，两个工作表中的数据除了 Date 列中的日期，其余是完全一样的。

(13) 将 Excel 文件保存在 file_archive 文件夹中，命名为 supplies.xls。
(14) 最后，这是一个可选步骤，如果你可以将文件保存成 Excel 工作簿格式（.xlsx），打开“Save As”对话框，将文件保存为 supplies.xlsx。

现在，在 file_archive 文件夹中，你应该有 3 个文件。

- CSV 文件：supplies_2012.csv
- Excel 文件：supplies.xls
- Excel 工作簿文件（可选）：supplies.xlsx

如果你没有创建可选的 .xlsx 文件，示例代码也可以工作，只是会缺少一些输出。这 3 个文件可以作为我们积累的历史文件，但是请记住，示例代码可以扩展为处理任意多的 CSV 文件和 Excel 文件，只要计算机能力足够[1]。如果你有成百上千个 CSV 或 Excel 历史文件，那么你仍然可以使用这个示例中的代码作为具体搜索问题的起点，然后扩展代码。

现在我们已经有了用来搜索记录的文件夹和文件，还需要以某种方式来识别要搜索的记录。例如，我们要搜索特定的数值项目。如果只想搜索很少的数值项目，就可以使用列表或元组变量在 Python 脚本中将条件写死（例如：`items_to_look_for = ['1234', '2345']`），但是当要搜索的数值项目增加时，这种方法就会变得笨重甚至不可行。因此，我们要使用以前向脚本中传递输入数据的方法，并将数值项目放在 CSV 输入文件的一列中。使用这种方法，如果你想搜索几十、几百甚至几千个数值项目，都可以将它们写在 CSV 输入文件中，然后将这些输入数据读入 Python 脚本。这种输入方法具有很好的扩展性，特别是与将条件写死在 Python 脚本中的方法相比。

要列出识别搜索记录的数值项目：

(1) 打开 Excel，输入图 5-5 中的数据；
(2) 将文件保存为 item_numbers_to_find.csv。

注 1：计算机能处理多少文件主要取决于随机存取存储器（RAM）和中央处理单元（CPU）。Python 将要处理的数据保存在 RAM 中，所以当数据体积大于计算机 RAM 时，计算机就会将数据写在磁盘中，而不是 RAM 中。向磁盘中写数据的速度要明显慢于在 RAM 中存取数据，所以计算机会变得非常慢，甚至看上去没有反应。根据你的数据量，如果你觉得会遇到这种问题，就应该使用有更多 RAM 的机器，或者向机器中加入更多的 RAM，或者将数据分成小块来处理。你也可以使用分布式系统，将很多计算机连接在一起作为一个整体使用，但是这已经超出了本书范围。

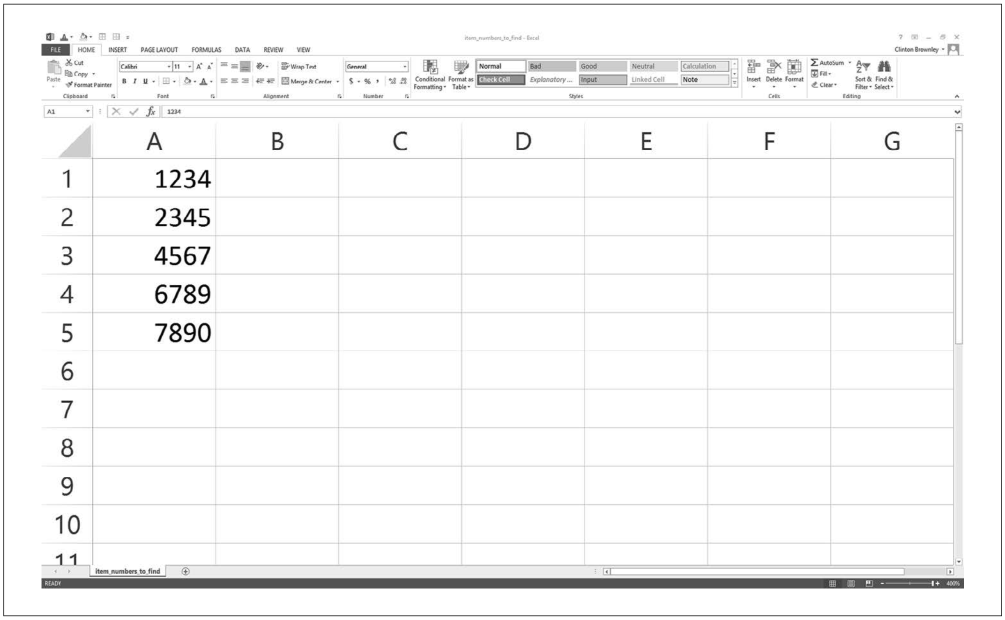


图 5-5：item_numbers_to_find.csv 中的示例数据，显示在 Excel 工作表中

你可以看到，我们要搜索的 5 个数值项目是 1234、2345、4567、6789 和 7890。它们都被写在第 A 列，没有标题行。其实可以包括标题行，但是没有必要，因为我们知道要使用哪一列，并且知道这列数值的意义。而且，如果添加了标题行，还需要多写一些代码来删除它，因为我们不会搜索输入文件标题行中的数据。如果将来有人或系统提供给你的数值列表中包括标题行，那么你可以使用在前面的章节中学会的去掉标题行的方法，将其读入一个变量，然后不使用这个变量就可以了。如果你自己创建列表，就应该不包括标题行，这样做可以简化进行数据处理的代码，你可以根据文件名称和文件所在的项目文件夹名称回想起数据的意义。

至此，我们已经理解了搜索任务，并且创建了执行示例代码所需的文件和文件夹。简而言之，我们的任务是搜索 file_archive 文件夹，找出包含我们所需的数值项目的文件，当找到一个数值项目时，需要把包含这个项目的整行数据写入输出文件。通过这种方式，你可以在历史信息中找出所有可以用于与供应商谈判的信息，这些信息都与数值项目相关。我们要搜索 3 个历史文件：一个 CSV 文件、一个 Excel 文件（.xls）和一个 Excel 工作簿文件（.xlsx）。这 3 个文件构成了示例代码能够处理的文件的最小集，脚本代码可以扩展为处理任意数量的文件，只要计算机能力足够。我们还建立了一个独立的 CSV 文件，包含要搜索的数值项目。在这个文件中，可以包含成百上千甚至更多的数值项目，这种输入方法同样可以帮助我们扩展搜索能力。

我们已经创建了 file_archive 文件夹和所有输入文件，现在需要做的就是编写 Python 代码来执行搜索任务。要完成这个操作，在文本编辑器中输入下列代码，然后将文件保存为 1search_for_items_write_found.py：

```python
#!/usr/bin/env python3
import csv
import glob
import os
import sys
from datetime import date
from xlrd import open_workbook, xldate_as_tuple
item_numbers_file = sys.argv[1]
path_to_folder = sys.argv[2]
output_file = sys.argv[3]
item_numbers_to_find = []
with open(item_numbers_file, 'r', newline='') as item_numbers_csv_file:
    filereader = csv.reader(item_numbers_csv_file)
    for row in filereader:
         item_numbers_to_find.append(row[0])
#print(item_numbers_to_find)
filewriter = csv.writer(open(output_file, 'a', newline=''))
file_counter = 0
line_counter = 0
count_of_item_numbers = 0
for input_file in glob.glob(os.path.join(path_to_folder, '*.*')):
    file_counter += 1
    if input_file.split('.')[1] == 'csv':
         with open(input_file, 'r', newline='') as csv_in_file:
              filereader = csv.reader(csv_in_file)
              header = next(filereader)
              for row in filereader:
                   row_of_output = [ ]
                   for column in range(len(header)):
                         if column == 3:
                              cell_value = str(row[column]).lstrip('$').\
                              replace(',','').strip()
                              row_of_output.append(cell_value)
                         else:
                              cell_value = str(row[column]).strip()
                              row_of_output.append(cell_value)
                  row_of_output.append(os.path.basename(input_file))
                  if row[0] in item_numbers_to_find:
                       filewriter.writerow(row_of_output)
                       count_of_item_numbers += 1
                  line_counter += 1
    elif input_file.split('.')[1] == 'xls' or \
    input_file.split('.')[1] == 'xlsx':
         workbook = open_workbook(input_file)
         for worksheet in workbook.sheets():
              try:
                   header = worksheet.row_values(0)
              except IndexError:
                   pass
              for row in range(1, worksheet.nrows):
                   row_of_output = [ ]
                   for column in range(len(header)):
                        if worksheet.cell_type(row, column) == 3:
                             cell_value = \
```

```
55                             xldate_as_tuple(worksheet.cell(row,column)\
56                             .value,workbook.datemode)
57                             cell_value = str(date(*cell_value[0:3])).strip()
58                             row_of_output.append(cell_value)
59                         else:
60                             cell_value = \
61                             str(worksheet.cell_value(row,column)).strip()
62                             row_of_output.append(cell_value)
63                     row_of_output.append(os.path.basename(input_file))
64                     row_of_output.append(worksheet.name)
65                     if str(worksheet.cell(row,0).value).split('.')[0].strip() \
66                     in item_numbers_to_find:
67                         filewriter.writerow(row_of_output)
68                         count_of_item_numbers += 1
69                     line_counter += 1
70 print('Number of files:', file_counter)
71 print('Number of lines:', line_counter)
72 print('Number of item numbers:', count_of_item_numbers)
```

这个脚本比前面章节中的任何一个脚本都要长，但是如果你完成了前面章节中的示例代码，那么应该对这个脚本中的所有代码都很熟悉了。第 2~7 行代码导入我们读取和处理输入数据所需的模块和方法。这里需要导入 csv、glob、os、string 和 sys 模块，来分别读写 CSV 文件、读取一个文件夹中的多个文件、在一个特定路径中搜索文件、处理字符串变量和在命令行中输入文件名。像在第 3 章中一样，这里还要导入 datetime 模块的 date 方法和 xlrd 模块的 xldate_as_tuple 方法，确保我们从输入文件中提取的任何日期数据都能以特定的形式保存到输出文件中。

第 8、9、10 行代码读取我们在命令行中提供的 3 个输入参数，分别是包含要搜索的数值项目的 CSV 文件的路径名、包含要搜索的文件的 file_archive 文件夹的路径、包含在历史文件中搜索到的与数值项目相关的信息行的 CSV 输出文件的路径名。这里将这 3 个输入参数赋给 3 个独立的变量，分别是 item_numbers_file、path_to_folder、和 output_file。

要在代码中使用我们想要搜索的数值项目，需要将它们从 CSV 输入文件转换成合适的数据结构，比如一个列表。第 11~15 行代码完成了这个转换。第 11 行代码创建了一个空列表 item_numbers_to_find。第 12 和 13 行代码使用 csv 模块的 reader() 方法打开 CSV 输入文件，并创建了一个 filereader 对象读取文件中的数据。第 14 行代码创建了一个 for 循环，在输入文件的所有行中循环。第 15 行代码使用列表的 append() 方法为在第 11 行中创建的列表添加值。为列表添加的值来自于 CSV 输入文件中的第一列 row[0]。如果你想看一下追加到列表中的数值项目，想在运行脚本时将其打印到屏幕上，那么就可以将第 16 行代码中 print 语句前面的注释符号去掉。

第 17 行代码使用 csv 模块的 write() 方法以追加方式（'a'）打开一个 CSV 输出文件，并创建一个 filewriter 对象，准备写入数据到输出文件。

第 18、19 和 20 行代码创建了 3 个计数变量，来跟踪（a）读入脚本的历史文件数量，（b）在所有的输入文件和工作表中读出的行数，和（c）行中数值项目是我们要搜索的数值项目的行数。这 3 个计数变量都初始化为 0。

第 21 行代码是外部 for 循环，在历史文件夹中的所有输入文件中循环。这行代码使用 os.path.join() 函数和 glob.glob() 函数来在 file_archive 文件夹中找到所有匹配于一个特定模式的文件。file_archive 文件夹的路径由我们在命令行中提供，包含在变量 path_to_folder 中。os.path.join() 函数将这个文件夹路径与文件夹中所有文件的文件名连接起来，这些匹配于特定模式的文件名由 glob.glob() 函数进行扩展。这里，我们使用模式 '*.*' 来匹配以任意扩展名结尾的任意文件名。在这个例子中，因为我们创建了输入文件夹和文件，所以知道文件夹中的文件扩展名只有 .csv、.xls 和 .xlsx。如果你只想搜索 CSV 文件，那么可以使用 '*.csv'；如果你只想搜索 .xls 或 .xlsx 文件，那么可以使用 '*.xls*'。这是一个 for 循环，所以这行代码的其他语法就很熟悉了。input_file 是一个占位符名称，代表由 glob.glob() 函数生成的列表中的每个文件。

第 22 行代码对于读入脚本的每一个输入文件，都将 file_counter 变量的值增加 1。在所有输入文件都被读入脚本之后，file_counter 就是读入的输入文件的总数。

第 23 行代码是一个 if 语句，开始了一个处理 CSV 文件的代码块。与这行代码同级的是第 42 行中的 elif 语句，它开始了一个处理 .xls 和 .xlsx 文件的代码块。第 23 行代码使用 string 模块的 split() 方法将每个输入文件的路径名按照路径中的句点（.）进行分割。例如，CSV 输入文件的路径名是 file_archive\supplies_2012.csv。如果这个字符串按照句点分割，句点前面部分的索引值为 [0]，句点后面的索引值为 [1]。这行代码检验句点后面的字符串（索引值为 1）是否为 csv，对于 CSV 输入文件这行代码为真。因此，第 24~41 行代码仅对于 CSV 输入文件执行。

第 24 和 25 行代码我们已经非常熟悉了。它们使用 csv 模块的 reader() 方法打开 CSV 输入文件，并创建了一个 filereader 对象，从文件中读取数据。

第 26 行代码使用 next() 方法读取输入文件中的第一行数据，也就是标题行，并赋给变量 header。

第 27 行代码创建了一个 for 循环，在 CSV 文件中余下的数据行之间循环。对于每一行，如果这行包含我们要搜索的数值项目，那么我们就需要组装一行输出写入输出文件。为了准备组装这行输出，第 28 行代码创建了一个空列表变量 row_of_output。

第 29 行代码创建了一个 for 循环，在输入文件一个给定行的各列之间循环。这行代码使用 range() 函数和 len() 函数创建了一个 CSV 输入文件各列的索引列表。因为输入文件中有 5 列，所以 column 变量的范围是从 0 到 4。

第 30~36 行代码包含一个 if-else 语句，对不同列中的值进行不同的处理。if 代码块处理索引值为 3 的列，也就是第四列 Cost。对于这一列，先使用 lstrip() 方法剥离字符串左侧的美元符号，然后使用 replace() 方法用空字符串替换掉字符串中的逗号（这样可以有效地删除逗号），再使用 strip() 方法剥离字符串两端的空格、制表符和换行符。在这些处理完成之后，第 33 行代码将最后的值追加到列表 row_of_output 中。

else 代码块处理其余各列中的数据。对于这些值，使用 strip() 方法剥离字符串两端的空格、制表符和换行符，然后让结果追加到第 36 行的列表 row_of_output 中。

第 37 行代码将输入文件的基础文件名追加到列表 row_of_output 中。对于 CSV 文件，变量 input_file 中包含的字符串是 file_archive\supplies_2012.csv。os.path.basename 确

保代码只将 supplies_2012.csv 追加到列表 row_of_output 中。

至此，CSV 输入文件中的第一行数据被读入到脚本中，这行数据中的每一列都进行了处理，然后追加到列表 row_of_output 中。现在应该检验这一行中的数值项目是否就是我们要搜索的数值项目。第 38 行代码进行了这个判断。这行代码检验这一行数据中第一列的值（是个数值项目）是否在我们要搜索的数值项目列表中，即是否包含在列表变量 item_numbers_to_find 中。如果这个数值项目是我们要搜索的数值项目中的一个，那么我们就在第 39 行中使用 filewriter 的 writerow() 方法，将这一行写入 CSV 输出文件。在第 40 行代码中，我们还要将 count_of_item_numbers 变量的值增加 1，来跟踪在所有输入文件中找到的数值项目的数量。

最后，在转到处理 CSV 输入文件中的下一行数据之前，在第 41 行代码中，要将 line_counter 变量的值增加 1，来跟踪我们在所有输入文件中找到的数据行的数量。

从第 42 行开始到第 69 行是另一个代码块，它与前一个代码块非常相似，区别在于它处理的是 Excel 文件（.xls 和 .xlsx），不是 CSV 文件。因为这个“Excel”代码块中的逻辑与“CSV”代码块中的逻辑是一样的，唯一区别是处理 Excel 文件的语法和 CSV 文件不一样，所以这里就不像前面那样详细地解释每行代码了。

第 42 行是一个 elif 语句，开始了一个处理扩展名为 .xls 或 .xlsx 的 Excel 文件的代码块。第 42 行使用一个“or”条件来检验文件扩展名是否为 .xls 或 .xlsx。因此，第 43~69 行代码执行的是扩展名为 .xls 和 .xlsx 的 Excel 输入文件。

第 43 行代码使用 xlsx 模块的 open_workbook() 方法打开一个 Excel 工作簿，并将其中的内容赋给变量 workbook。

第 45 行代码创建了一个 for 循环，在工作簿的所有工作表之间循环。对于每个工作表，第 46~49 行代码先试图将工作表中的第一行（也就是标题行）读入变量 header。如果出现了 IndexError，就说明序列下标越界，那么就执行 Python 关键字 pass，实际上就是什么都不做，然后代码继续执行第 50 行操作。

第 50 行代码创建了一个 for 循环，在 Excel 输入文件的其余数据行之间循环。循环范围不是从 0 开始，而是从 1 开始，也就是从工作表中的第二行开始（成功地跳过了标题行）。

这个“Excel”代码块中其余各行代码与“CSV”代码块中的代码基本相同，除了使用 Excel 解析语法代替了 CSV 解析语法。if 代码块处理单元格类型为 3 的列，也就是包含代表日期的数值的列。这个代码块使用 xlrd 模块的 xldate_as_tuple() 方法和 datetime 模块的 date() 方法来保证这个列中的日期值在输出文件中保持原来的格式。只要这个值被转换为具有日期形式的文本字符串，就使用 strip() 方法剥离字符串两端的空格、制表符和换行符，然后第 58 行代码使用列表的 append() 方法将这个值追加到列表 row_of_output 中。

else 代码块处理所有其他列中的值。对于每个值，使用 strip() 方法剥离字符串两端的空格、制表符和换行符，然后第 62 行代码将这个值追加到列表 row_of_output 中。

第 63 行代码将输入文件的基本文件名追加到列表 row_of_output 中。与 CSV 文件不同，Excel 文件中可以包含多个工作表。因此，第 64 行代码还将工作表名称追加到列表中。这个 Excel 文件的附加信息可以更加清楚地表示出脚本在哪里找到了数值项目。

第 65~69 行代码和处理 CSV 文件的代码一样，第 65 行检验行中第一列的值（数值项目）是否在我们要搜索的数值项目列表中，即是否包含在列表变量 `item_numbers_to_find` 中。如果数值项目是我们要搜索的数值项目中的一个，那么就在第 67 行中使用 `filewriter` 的 `writerow()` 方法将这一行写入 CSV 输出文件。在第 68 行代码中，我们还要将 `count_of_item_numbers` 变量的值增加 1，来跟踪在所有输入文件中找到的数值项目的数量。

最后，在转到处理 Excel 工作表中的下一行数据之前，在第 69 行代码中，要将 `line_counter` 变量的值增加 1，来跟踪我们在所有输入文件中找到的数据行的数量。

第 70、71 和 72 行代码是 `print` 语句。当脚本处理完成所有输入文件时，这些 `print` 语句将摘要信息打印在命令行窗口或终端窗口中。第 70 行打印出脚本处理的文件数。第 71 行打印出在所有输入文件和工作表中读取的行数。第 72 行打印出带有我们要搜索的数值项目的行数，这个数值可能包含重复计数。例如，如果数值项目“1234”在一个文件中出现了两次或者在两个文件中分别出现一次，那么这个项目在这行代码打印到命令行窗口或终端窗口中的数值中就被计算了两次。

现在我们已经完成了 Python 脚本，可以使用这个脚本在一堆历史文件中搜索特定的数据行，并将输出写入一个 CSV 格式的输出文件了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 1search_for_items_write_found.py item_numbers_to_find.csv file_archive\
output_files\1app_output.csv
```

在 Windows 系统中，你可以看到输出被打印到命令行窗口中，如图 5-6 所示。

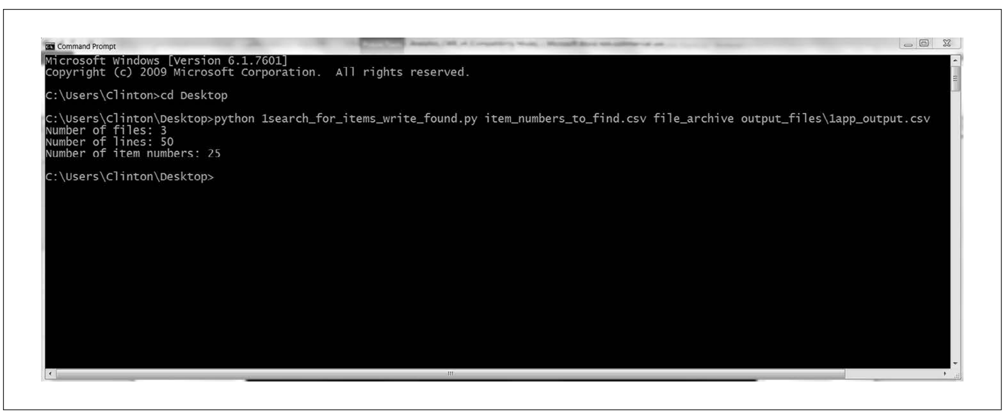
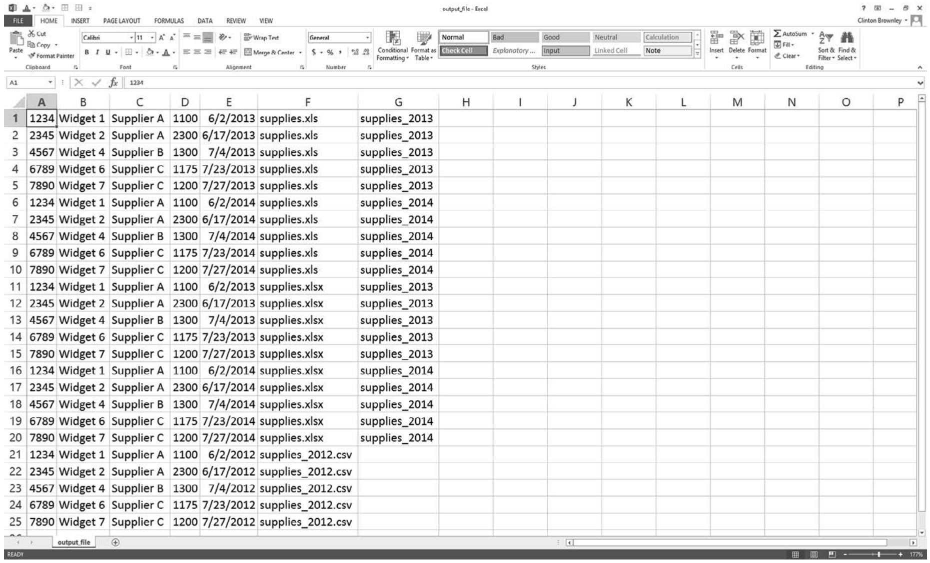


图 5-6：1search_for_items_write_found.py 的运行结果，使用 item_numbers_to_find.csv 文件和 file_archive 文件夹中的文件

从命令行窗口中的输出你可以看到，脚本处理了 3 个输入文件，从输入文件中读入了 50 行数据，并找到了 25 行包含我们需要的数值项目的数据。这个输出没有表示出脚本找到了多少个数值项目，也没有表示出每个数值项目被找到了多少次。但是，这就是我们要将输出写在 CSV 输出文件中的原因。

要查看输出文件内容，需要打开文件 1app_output.csv。内容如图 5-7 所示。



	A	B	C	D	E	F	G
1	1234	Widget 1	Supplier A	1100	6/2/2013	supplies.xls	supplies_2013
2	2345	Widget 2	Supplier A	2300	6/17/2013	supplies.xls	supplies_2013
3	4567	Widget 4	Supplier B	1300	7/4/2013	supplies.xls	supplies_2013
4	6789	Widget 6	Supplier C	1175	7/23/2013	supplies.xls	supplies_2013
5	7890	Widget 7	Supplier C	1200	7/27/2013	supplies.xls	supplies_2013
6	1234	Widget 1	Supplier A	1100	6/2/2014	supplies.xls	supplies_2014
7	2345	Widget 2	Supplier A	2300	6/17/2014	supplies.xls	supplies_2014
8	4567	Widget 4	Supplier B	1300	7/4/2014	supplies.xls	supplies_2014
9	6789	Widget 6	Supplier C	1175	7/23/2014	supplies.xls	supplies_2014
10	7890	Widget 7	Supplier C	1200	7/27/2014	supplies.xls	supplies_2014
11	1234	Widget 1	Supplier A	1100	6/2/2013	supplies.xlsx	supplies_2013
12	2345	Widget 2	Supplier A	2300	6/17/2013	supplies.xlsx	supplies_2013
13	4567	Widget 4	Supplier B	1300	7/4/2013	supplies.xlsx	supplies_2013
14	6789	Widget 6	Supplier C	1175	7/23/2013	supplies.xlsx	supplies_2013
15	7890	Widget 7	Supplier C	1200	7/27/2013	supplies.xlsx	supplies_2013
16	1234	Widget 1	Supplier A	1100	6/2/2014	supplies.xlsx	supplies_2014
17	2345	Widget 2	Supplier A	2300	6/17/2014	supplies.xlsx	supplies_2014
18	4567	Widget 4	Supplier B	1300	7/4/2014	supplies.xlsx	supplies_2014
19	6789	Widget 6	Supplier C	1175	7/23/2014	supplies.xlsx	supplies_2014
20	7890	Widget 7	Supplier C	1200	7/27/2014	supplies.xlsx	supplies_2014
21	1234	Widget 1	Supplier A	1100	6/2/2012	supplies_2012.csv	
22	2345	Widget 2	Supplier A	2300	6/17/2012	supplies_2012.csv	
23	4567	Widget 4	Supplier B	1300	7/4/2012	supplies_2012.csv	
24	6789	Widget 6	Supplier C	1175	7/23/2012	supplies_2012.csv	
25	7890	Widget 7	Supplier C	1200	7/27/2012	supplies_2012.csv	

图 5-7：1search_for_items_write_found.py 写入到 1app_output.csv 中的数据

这些记录来自于 3 个输入文件中的行数据，其中的数值项目都与 CSV 文件中的数值项目相匹配。最后第二列给出了数据所在的文件名。如果数据包含在这两个 Excel 工作簿中，最后一列就给出了数据所在的工作表名称。

从输出文件内容可以看出，我们找到了 25 行包含我们需要的数值项目的数据。这个输出文件与命令行窗口中打印出的“25”是一致的。特别需要注意的是，我们发现每个数值项目在所有输入文件中都一共被找到了 5 次。例如，数值项目“1234”在 .xls 文件中被找到了 2 次（一次在 supplies_2013 工作表中，一次在 supplies_2014 工作表中），在 .xlsx 文件中被找到了 2 次（一次在 supplies_2013 工作表中，一次在 supplies_2014 工作表中），在 CSV 输入文件中被找到了 1 次。

与来自于 CSV 输入文件的行相比，来自于 Excel 工作表的行中多出了一列（就是找到数据行的工作表名称）。第四列中的成本数据只包括输入文件中的成本数值部分。最后，第五列中的日期数据已经被格式化了，与 CSV 和 Excel 输入文件中的格式保持一致。

这个应用程序综合运用了我们在前几章中学习的技术，解决了一个常见的实际问题。商业分析师们经常会遇到需要将分布在多个不同类型文件中的历史数据组合成一个完整数据集这样的问题。在很多情况下，历史数据文件会有几十、几百甚至上千个，从这些文件中搜索并提取出特定数据想想就令人望而生畏。

这一节演示了一种可扩展的从历史记录集合中提取特定记录的方法。为了使这个示例尽量简单，这里仅仅使用了很少的数值项目和 3 个历史记录文件。但是，这种方法具有很好的扩展性，你可以在更多的数值项目和更大的历史文件集合中使用这种方法。

至此，我们已经解决了在一个大的历史文件集合中搜索特定记录的问题。下面开始解决为一个未知数目的分类计算统计量的问题。这个目标此刻听起来有点抽象，我将在下一节对这个问题进行详细讨论，并给出解决方法。

5.2 为CSV文件中数据的任意数目分类计算统计量

在很多商业分析中，需要为一个特定时间段内的未知数目的分类计算统计量。举例来说，假设我们销售 5 种不同种类的产品，你想计算一下在某一年中对于所有客户的按产品种类分类的总销售额。因为客户具有不同的品味和偏好，他们在一年中购买的产品也是不同的。有些客户购买了所有 5 种产品，有些客户则只购买了一种产品。在这种客户购买习惯之下，与每个客户相关的产品分类数目都是不同的。

如果想简单处理的话，你可以为每个客户都分配全部 5 种产品分类，将所有产品分类的初始总销售额都设为 0，然后只将每个客户实际购买的产品计算到总销售额中。但是，我们已经知道了很多客户只购买一种或两种产品，而且你也只关心客户实际购买的产品的总销售额。为所有客户计算全部 5 种产品分类的销售额不仅没有必要，还是一种干扰，同时也是对内存、计算能力和存储空间的浪费。基于以上原因，我们应该只处理必要的数据，为每个客户计算他们购买的产品以及每种产品分类的总销售额。

再举个例子，假如客户对不同的产品或服务包的购买力会随着时间的推移而有所提高。例如，你向客户提供了铜牌、银牌和金牌 3 种类型的服务包。针对这 3 种类型，有些客户首选了铜牌服务包，有些客户首选了银牌服务包，也有些客户首选了金牌服务包。对于那些首选铜牌和银牌服务包的客户，随着时间的推移，他们可能会购买价值更高的产品与服务包。

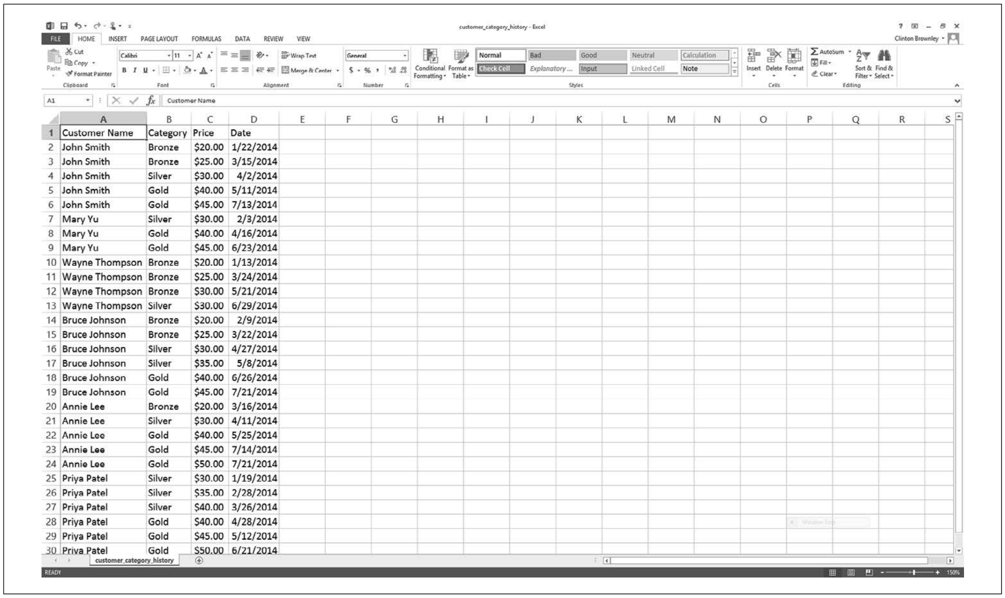
现在，你非常想计算出你的客户在他们购买的每个服务包类别上花费的总时间（以月计算）。例如，如果你的一个客户 Tony Shephard 在 2014 年 2 月 15 日购买了铜牌服务包，在 2014 年 6 月 15 日购买了银牌服务包，在 2014 年 9 月 15 日购买了金牌服务包，那么关于 Tony Shephard 的计算结果就是："铜牌服务包：4 个月""银牌服务包：3 个月""金牌服务包：从 2014 年 9 月 15 日至今"。如果另一个客户 Mollie Adler 只购买了银牌服务包和金牌服务包，那么关于 Mollie Adler 的计算结果中就不会包含铜牌服务包的任何信息。

如果你的客户数据集非常小，那么你可以打开文件，计算出日期之间的差额，然后按照服务包类别和客户名称将它们累加起来。但是，这种手工处理方法不但浪费时间，还容易出错。而且，当文件太大难以打开时，就不好办了。这就是使用 Python 的绝好机会。Python 可以处理因体积太大而难以打开的文件，它的计算速度非常快，而且能够减少人为出错的机会。

为了在一个客户购买服务包的数据集上执行计算，需要先创建一个 CSV 数据文件。

(1) 打开 Excel，输入图 5-8 所示的数据。

(2) 将文件保存为 customer_category_history.csv。



Customer Name	Category	Price	Date
John Smith	Bronze	$20.00	1/22/2014
John Smith	Bronze	$25.00	3/15/2014
John Smith	Silver	$30.00	4/2/2014
John Smith	Gold	$40.00	5/11/2014
John Smith	Gold	$45.00	7/13/2014
Mary Yu	Silver	$30.00	2/3/2014
Mary Yu	Gold	$40.00	4/16/2014
Mary Yu	Gold	$45.00	6/23/2014
Wayne Thompson	Bronze	$20.00	1/13/2014
Wayne Thompson	Bronze	$25.00	3/24/2014
Wayne Thompson	Bronze	$30.00	5/21/2014
Wayne Thompson	Silver	$30.00	6/29/2014
Bruce Johnson	Bronze	$20.00	2/9/2014
Bruce Johnson	Bronze	$25.00	3/22/2014
Bruce Johnson	Silver	$30.00	4/27/2014
Bruce Johnson	Silver	$35.00	5/8/2014
Bruce Johnson	Gold	$40.00	6/26/2014
Bruce Johnson	Gold	$45.00	7/21/2014
Annie Lee	Bronze	$20.00	3/16/2014
Annie Lee	Silver	$30.00	4/11/2014
Annie Lee	Gold	$40.00	5/25/2014
Annie Lee	Gold	$45.00	7/14/2014
Annie Lee	Gold	$50.00	7/21/2014
Priya Patel	Silver	$30.00	1/19/2014
Priya Patel	Silver	$35.00	2/28/2014
Priya Patel	Silver	$40.00	3/26/2014
Priya Patel	Gold	$40.00	4/28/2014
Priya Patel	Gold	$45.00	5/12/2014
Priya Patel	Gold	$50.00	6/21/2014

图 5-8：customer_category_history.csv 中的示例数据，显示在 Excel 工作表中

你可以看出，这个数据集包括 4 列数据：Customer Name、Category、Price 和 Date。还包括 6 个客户：John Smith、Mary Yu、Wayne Thompson、Bruce Johnson、Annie Lee 和 Priya Patel。同时包括 3 个服务包分类：铜牌、银牌和金牌。数据是以先按照客户姓名，再按照日期的形式升序排列的。

现在我们已经有了数据集，其中包括客户在过去一年中购买的服务包，还有服务包的购买日期或更新日期。接下来要做的就是编写 Python 代码来执行计算。

要完成这个操作，在文本编辑器中输入下列代码，然后将文件保存为 2calculate_statistic_by_category.py：

```
#!/usr/bin/env python3
import csv
import sys
from datetime import date, datetime

def date_diff(date1, date2):
    try:
        diff = str(datetime.strptime(date1, '%m/%d/%Y') - \
        datetime.strptime(date2, '%m/%d/%Y')).split()[0]
    except:
        diff = 0
    if diff == '0:00:00':
        diff = 0
    return diff
input_file = sys.argv[1]
output_file = sys.argv[2]
packages = { }
```

```
previous_name = 'N/A'
previous_package = 'N/A'
previous_package_date = 'N/A'
first_row = True
today = date.today().strftime('%m/%d/%Y')
with open(input_file, 'r', newline='') as input_csv_file:
    filereader = csv.reader(input_csv_file)
    header = next(filereader)
    for row in filereader:
        current_name = row[0]
        current_package = row[1]
        current_package_date = row[3]
        if current_name not in packages:
            packages[current_name] = { }
        if current_package not in packages[current_name]:
            packages[current_name][current_package] = 0
        if current_name != previous_name:
            if first_row:
                first_row = False
            else:
                diff = date_diff(today, previous_package_date)
                if previous_package not in packages[previous_name]:
                    packages[previous_name][previous_package] = int(diff)
                else:
                    packages[previous_name][previous_package] += int(diff)
        else:
            diff = date_diff(current_package_date, previous_package_date)
            packages[previous_name][previous_package] += int(diff)
        previous_name = current_name
        previous_package = current_package
        previous_package_date = current_package_date
header = ['Customer Name', 'Category', 'Total Time (in Days)']
with open(output_file, 'w', newline='') as output_csv_file:
    filewriter = csv.writer(output_csv_file)
    filewriter.writerow(header)
    for customer_name, customer_name_value in packages.items():
        for package_category, package_category_value \
        in packages[customer_name].items():
            row_of_output = [ ]
            print(customer_name, package_category, package_category_value)
            row_of_output.append(customer_name)
            row_of_output.append(package_category)
            row_of_output.append(package_category_value)
            filewriter.writerow(row_of_output)
```

脚本中的代码完成了计算任务，同时也很有值得学习的意义。这是本书中第一个使用 Python 字典数据结构来组织和保存计算结果的示例。实际上，这个脚本中的示例比普通的字典还要复杂，因为它是一个嵌套的字典，也就是字典的字典。这个示例展示了创建字典和使用键-值对填充字典的便捷性。在这个示例中，外部字典的名称为 packages。外部字典的键为客户名称，与这个键相对的值是另一个字典，其中的键为服务包类别的名称，值为一个整数，表示客户拥有这个服务包的天数。字典是一种方便易懂的数据结构，因为很多数据源和分析技术都支持这种键-值对结构。你可以回忆一下，在第 1 章中，字典是由

花括号（{}）创建的，字典中的键是唯一的字符串，键–值对中的键和值由冒号分隔，每个键–值对之间由逗号分隔。例如，costs={'people':3640, 'hardware':3975}。

其次，这个脚本还演示了如何对一个具体类别中的第一行数据进行特别的处理，这种处理与这个类别中其余各行是不同的，因为我们要在两行之差的基础上计算统计量。举例来说，在脚本中，在外层 if 语句 if current_name != previous_name 中的所有代码仅用来处理一个新客户的第一行数据，其余各行客户数据都使用外层的 else 语句进行处理。

最后，这个脚本演示了如何定义和使用用户自定义函数。这个脚本中的函数 date_diff 计算并返回两个日期之间间隔的天数。这个函数在第 6~14 行代码中定义，在第 38 和 44 行代码中被调用。如果我们没有定义函数，那么函数中的代码就会在脚本中重复输入两次，第一次在第 38 行，第二次在第 44 行。通过定义函数，你只需要将这些代码输入一次，这样不但可以减少脚本中的代码行数，还可以简化第 38 行和第 44 行代码。正如第 1 章中提到的那样，只要发现在脚本中有重复的代码，就可以考虑将这些代码打包成一个函数，并使用这个函数来简化和缩短脚本中的代码。

我们已经介绍了脚本中需要注意的几个方面，下面来逐行讨论代码。第 2~5 行代码导入读取和处理数据所需的模块和方法。我们导入了 csv、datetime、string 和 sys 模块，来分别读取和写入 CSV 文件、处理日期变量、处理字符串变量和在命令行中输入参数。我们从 datetime 模块导入 date 和 datetime 方法，来访问当前日期和计算日期之间的间隔。

第 6~14 行代码定义了一个用户自定义函数 date_diff。第 6 行代码包含了函数定义语句，它指定了函数名称，并说明这个函数具有两个参数 date1 和 date2。第 7~11 行代码包含了一个 try-except 异常处理语句。try 代码块试图使用 datetime.strptime() 函数按照日期字符串创建 datetime 对象，并用第一个日期减去第二个日期，使用 str() 函数将相减的结果转换成一个字符串，然后将结果字符串使用 split() 函数按照空格进行分割，最后保留字符串分割后最左边的部分（索引值为 [0] 的字符串），并将其赋给变量 diff。如果 try 代码块遇到异常，则执行 except 代码块。如果被执行，except 代码块就将 diff 的值设为整数 0。同样，第 12 和 13 行代码是一个 if 语句，处理当两个日期相等，二者的差为 0（被格式化为 '0:00:00'）的情况。如果二者的差为 0（请注意用两个等号表示相等），那么 if 语句就将 diff 的值设为整数 0。最后，在第 14 行代码中，函数返回包含在变量 diff 中的整数值。

第 15 和 16 行代码读入我们在命令行中提供的两个参数：CSV 输入文件的路径名和 CSV 输出文件的路径名。CSV 输入文件中包含客户数据，CSV 输出文件中包含客户相关信息，以及他们拥有具体服务包的时间。这两个输入参数被分别赋给两个变量 input_file 和 output_file。

第 17 行代码创建了一个空字典 packages，用来保存我们需要的信息。第 18、19 和 20 行代码创建了 3 个变量：previous_name、previous_package 和 previous_package_date，并将一个字符串 'N/A' 赋给了每个变量。我们将 'N/A' 赋给这些变量有个假设前提，那就是字符串 'N/A' 不会出现在输入文件的客户姓名、服务包类别或服务包日期这 3 列中。如果你想根据自己的分析来修改代码，并且想让作为字典键的列中包括字符串 'N/A'，那么就应该将 'N/A' 修改为一个不会出现在输入文件各列中的字符串，比如 'QQQQQ' 或其他有特点的又没有实际意义的字符串。

第 21 行代码创建了一个布尔变量 first_row，并赋值为 True。我们使用这个变量来确定是否在处理输入文件中的第一行数据。如果正在处理第一行数据，那么就使用某个代码块来进行处理。如果正在处理的不是第一行数据，那么就使用另一个代码块来进行处理。

第 22 行代码创建了一个变量 today，包含当前日期，形式为 %m/%d/%Y。在这种日期形式下，2014 年 10 月 21 日显示为 10/21/2014。

第 23 和 24 行代码使用 with 语句和 csv 模块的 reader 方法打开 CSV 输入文件，并创建一个 filereader 对象来读取文件中的数据。第 25 行代码对 filereader 对象使用 next 方法，从输入文件中读出第一行数据，并将其赋给变量 header。

第 26 行代码创建了一个 for 循环，在输入文件的其余数据行之间循环。第 27 行代码取出第一列的值 row[0]，并将其赋给变量 current_name。第 28 行代码取出第二列的值 row[1]，并将其赋给变量 current_package。第 29 行代码取出第四列的值 row[3]，并将其赋给变量 current_package_date。第一个数据行中包含的值为 John Smith、Bronze 和 1/22/2014，所以这些值被分别赋给变量 current_name、current_package 和 current_package_date。

第 30 行代码创建了一个 if 语句，用来检验 current_name 变量中的值是否还不是字典 packages 中的一个键。如果不是，那么第 31 行代码将 current_name 中的值作为字典键加入到字典 packages 中，并将与这个键对应的值设为一个空字典。这两行代码使用键-值对来填充字典 packages。

同样，第 32 行代码也创建了一个 if 语句，用来检验 current_package 变量中的值是否还不是内部字典中的一个键，这个内部字典是 packages 字典的一个值，对应的键为 current_name。如果不是，就使用第 33 行代码将 current_package 作为字典键添加到内部字典中，并将与其对应的值设为整数 0。这两行代码使用键-值对来填充与每个用户名称相关的内部字典。

举例来说，在第 31 行代码中，John Smith 成为 packages 字典中的一个键，与之对应的值为一个空字典。在第 33 行代码中，第一个与 John Smith 相关的服务包类别（就是铜牌服务包 Bronze）成为内部字典的键，与 Bronze 键对应的值被设为 0。这时，字典 packages 的内容就是 {'John Smith':{'Bronze':0}}。

第 34 行代码创建了一个 if 语句，检验变量 current_name 中的值是否不等于变量 previous_name 中的值。当第一次执行这行代码时，current_name 中的值为输入文件中第一个客户名称（也就是 John Smith）。previous_name 中的值为 'N/A'。因为 John Smith 不等于 'N/A'，所以我们进入 if 代码块。

第 35 行代码创建了一个 if 语句，用来检验代码是否正在处理输入文件的第一行数据。因为 first_row 变量的当前值为 True，所以执行第 36 行代码，将 first_row 的值设为 False。

然后，代码来到第 46、47 和 48 行，将 current_name、current_package 和 current_package_date 3 个变量的值分别赋给 previous_name、previous_package 和 previous_package_date 3 个变量。因此，previous_name 现在包含的值为 John Smith，previous_package 现在包含的值为 Bronze，previous_package_date 现在包含的值为 1/22/2014。

至此，脚本已经结束了对输入文件第一行数据的处理，所以脚本回到第 26 行代码，处理

文件中的下一行数据。对于这行数据，第 27、28 和 29 行代码分别将行中第一、第二和第四列数据赋给变量 current_name、current_package 和 current_package_date。因为第二行数据包含 John Smith、Bronze 和 3/15/2014，所以这些就是变量 current_name、current_package 和 current_package_date 中的值。

第 30 行代码再次检验 current_name 中的值是否还不是字典 packages 中的一个键。因为 John Smith 已经是字典中的键，所以第 31 行代码不被执行。同样，第 32 行代码再次检验变量 current_package 中的值是否还不是内部字典中的一个键。Bronze 已经是内部字典中的一个键，所以第 33 行代码不被执行。

此后，第 34 行代码检验 current_name 中的值是否等于 previous_name 中的值。current_name 中的值是 John Smith，previous_name 中的值也是 John Smith。因为这两个变量中的值相等，所以第 35~42 行代码被跳过，我们来到开始于第 43 行代码的 else 代码块。

第 44 行代码调用用户自定义函数 date_diff 来计算 current_package_date 变量值减去 previous_package_date 变量值的差，并将这个差（单位为天）赋给变量 diff。我们正在处理输入文件中的第二行数据，所以 current_package_date 中的值为 3/15/2014。在前一次循环中，我们将值 1/22/2014 赋给了变量 previous_package_date。因此，变量 diff 中的值就是 3/15/2014 减去 1/22/2014，也就是 52 天。

第 45 行代码将某个用户拥有某种服务包的时间加上变量 diff 的值。例如，循环到现在，previous_name 的值为 John Smith，previous_package 的值为 Bronze。因此，我们将 John Smith 拥有铜牌服务包的时间从 0 增加到 52 天。这时候，packages 字典中的内容就是：{'John Smith':{'Bronze':52}}，请注意其中的数值从 0 增加到了 52。

最后，第 46、47 和 48 行代码将 current_name、current_package 和 current_package_date 3 个变量的值分别赋给 previous_name、previous_package 和 previous_package_date 3 个变量。

为了确保你可以理解代码的工作方式，这里再讨论一次循环迭代。在前面一段中，我们知道 3 个 current_* 变量被赋给了 3 个 previous_* 变量。所以 previous_name、previous_package 和 previous_package_date 中的值分别为 John Smith、Bronze 和 3/15/2014。在循环的下一次迭代中，第 27、28 和 29 行代码将输入文件第三行数据中的值赋给 3 个 current_* 变量。在这次赋值之后，current_name、current_package 和 current_package_date 3 个变量中的值分别为 John Smith、Silver 和 4/2/2014。

第 30 行代码再次检验 current_name 中的值是否还不是字典 packages 中的一个键。因为 John Smith 已经是字典中的键，所以第 31 行代码不被执行。

第 32 行代码检验变量 current_package 中的值是否还不是内部字典中的一个键。这一次，current_package 中的值 Silver 是一个新值，还不是内部字典中的一个键，所以第 33 行代码将 Silver 作为内部字典的一个键，并将与 Silver 键对应的值初始化为 0。这个时候，packages 字典中的内容为 {'John Smith':{'Silver':0, 'Bronze':52}}。

此后，第 34 行代码检验 current_name 中的值是否等于 previous_name 中的值。current_name 中的值是 John Smith，previous_name 中的值也是 John Smith。因为这两个变量中的值相

等，所以第 35~42 行代码被跳过，我们来到开始于第 43 行代码的 else 代码块。

第 44 行代码调用用户自定义函数 date_diff 来计算 current_package_date 变量值减去 previous_package_date 变量值的差，并将这个差（单位为天）赋给变量 diff。因为我们正在处理输入文件中的第三行数据，所以 current_package_date 中的值为 4/2/2014。在前一次循环中，我们将值 3/15/2014 赋给了变量 previous_package_date。因此，变量 diff 中的值就是 4/2/2014 减去 3/15/2014，也就是 18 天。

第 45 行代码将某个用户拥有某种服务包的时间加上变量 diff 的值。例如，循环到现在，previous_name 的值为 John Smith，previous_package 的值为 Bronze。因此，我们将 John Smith 拥有铜牌服务包的时间从 52 天增加到 70 天。这时候，packages 字典中的内容就是：{'John Smith':{'Silver':0,'Bronze':70}}。

同样，第 46、47 和 48 行代码将 current_name、current_package 和 current_package_date 3 个变量的值分别赋给 previous_name、previous_package 和 previous_package_date 3 个变量。

当 for 循环处理完输入文件中所有行的时候，第 49~61 行代码将标题行和嵌套字典中的内容写入输出文件。第 49 行代码创建了一个列表变量 header，其中包含 3 个字符串：Customer Name、Category 和 Total Time (in Days)，这 3 个字符串将作为输出文件中 3 个列的标题。

第 50 和 51 行代码分别打开一个输出文件供脚本写入数据和创建一个写入对象来写入输出文件。第 52 行代码将 header 中的内容（也就是标题行）写入输出文件。

第 53 和 54 行代码是两个 for 循环，分别在外部字典和内部字典的键和值之间循环。外部字典的键是客户名称，与每个客户名称对应的值是另一个字典。内部字典的键是客户购买的服务包类别，内部字典的值是用户拥有的每个服务包的总时间（以天计）。

第 56 行代码创建了一个空列表 row_of_output，用来保存要写入输出文件的 3 个值。第 57 行代码将要输出的这 3 个值打印出来，这样我们就可以看到要写入输出文件中的内容了。当你确定脚本正确工作后，可以将这行代码删除。第 58~60 行代码将要输出的 3 个值追加到列表 row_of_output 中。最后，对于嵌套字典中的每一个客户名称和与之对应的服务包类别，第 61 行代码将我们需要的 3 个值以逗号分隔的形式写入输出文件。

我们已经完成了 Python 脚本，现在可以使用这个脚本计算每个顾客拥有不同服务类别的总时间，并将结果写入 CSV 输出文件了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 2calculate_statistic_by_category.py customer_category_history.csv\
output_files\2app_output.csv
```

你可以看到如图 5-9 所示的输出被打印在命令行窗口或终端窗口中（每人的实际数字会有不同，因为在脚本中使用的是当前日期）。

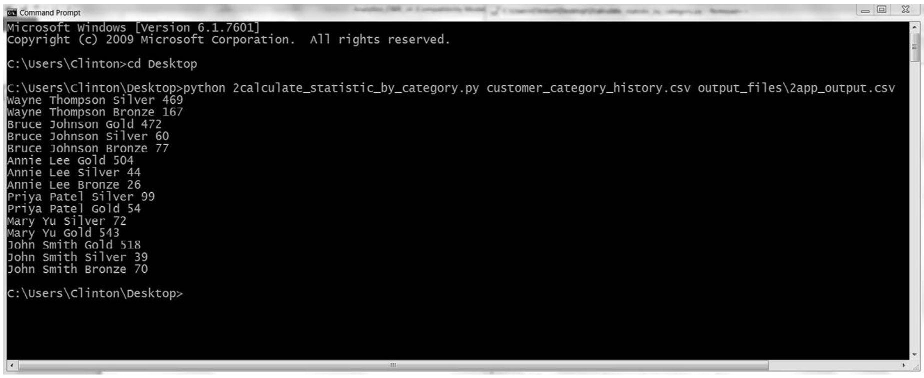


图 5-9：在 CSV 文件 customer_category_history.csv 上运行 2calculate_statistic_by_category.py 的结果

命令行窗口中的输出与写在 2app_output.csv 中的输出是一样的，这个文件中的内容如图 5-10 所示，它展示了每个客户拥有一个特定服务包的总天数。

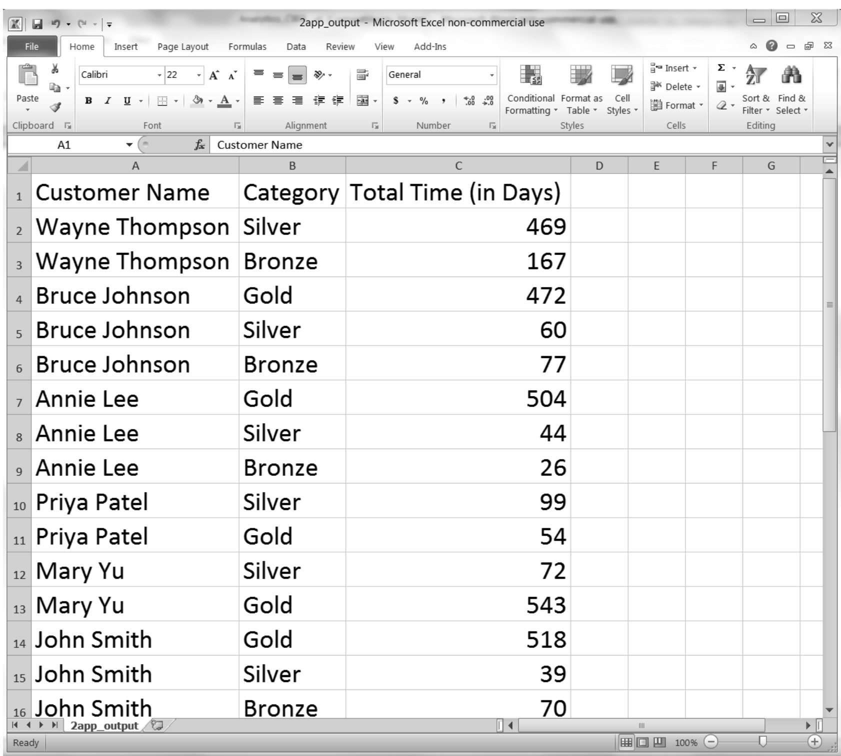


图 5-10：保存在 CSV 文件 2app_output.csv 中的脚本 2calculate_statistic_by_category.py 的输出（每个客户拥有一个特定服务包的总天数），显示在 Excel 工作表中

你可以看到，脚本先将标题行写入输出文件，然后写入了来自于输入文件的经过处理的信息，每行信息都以客户名称和服务包类别作为关键字。例如，标题行下面的前两个数据行说明了 Wayne Thompson 拥有铜牌服务包 167 天，银牌服务包 469 天。最后三行数据说明了 John Smith 拥有铜牌服务包 70 天，银牌服务包 39 天，金牌服务包 518 天。其余各行数据显示了其他客户的计算结果。对输入文件中的原始数据进行了处理和累加之后，你可以使用这些数据计算其他统计量，或者以各种方式进行摘要统计和数据可视化，或者将这些数据与其他数据结合起来进行更深入的分析。

还需要注意的一点是，对于如何处理每个客户的最后一种服务包类别，我们需要做出实际决策（或者说假设）。如果我们的输入数据是准确的而且是最新的，那么客户很可能还继续拥有上面的最后一个服务包。例如，Wayne Thompson 的最后一个服务包类别是银牌服务包，购买于 2014 年 6 月 29 日。因为 Wayne Thompson 很可能仍然拥有这种服务包，所以这个服务包的总时间应该是从 2014 年 6 月 29 日到今天为止的总时间，这说明每个客户拥有最后一个服务包的总时间取决于脚本在何时运行。

实现这个计算和累加的代码位于第 34 行下面的缩进部分。第 35 行代码保证在处理第一行数据时不进行减法计算，因为我们不能使用一个日期做减法。在处理完第一行数据之后，第 36 行代码将 `first_row` 设为 `False`。这样，对于所有其余的数据行，第 35 行代码都是 `False`，第 36 行代码不会被执行。对于每次从一个客户转换到另一个客户，第 38~42 行代码都要执行。第 38 行代码计算当前日期与 `previous_package_date` 之间的差值（以天计），并将其赋给变量 `diff`。然后，如果 `previous_package` 中的值还不是内部字典的键，第 40 行代码就将其作为内部字典的一个键，并将与之对应的值设为变量 `diff` 中的值。否则，如果 `previous_package` 中的值已经是内部字典的键，那么第 42 行代码就将 `diff` 中的值加在内部字典中与该键对应的字典值上面。

让我们回到 Wayne Thompson 的例子，Wayne Thompson 的最后一个服务包类别是银牌服务包，购买日期为 2014 年 6 月 29 日。下一行输入数据是另一个客户 Bruce Johnson，所以第 34 行下面的代码被执行。代码执行的时间就是我写这一章的时间，即 2015 年 10 月 11 日，所以 `diff` 中的值就是 10/11/2015 减去 6/29/2014 的差，也就是 469 天。在命令行窗口和输出文件中你都可以看到，Wayne Thompson 拥有银牌服务包的总时间是 469 天。

如果你在另一个时间运行了这个脚本，那么这行输出的总时间就会发生变化（应该是个更大的数字），同样，每个用户拥有最后服务包的时间也会发生变化。如何处理每个用户的最后一行数据要根据实际情况作出决策。你完全可以修改这个示例中的代码，以此来满足实际情况中的数据处理需求。

这个应用程序综合运用我们在第 1 章中学习的几种技术（例如创建用户自定义函数和填充字典）来解决一个常见的实际问题。商业分析师们会经常遇到这种需要计算输入数据中两行之间差值的问题。在很多情况下，需要以不同方式处理成千上万条数据，手动计算某些数据之间差值的想法简直是疯了（即使能够完成任务）。

这一节演示了一种可扩展的计算方法，可以用来计算行间的差值，并将这些差值按照输入文件中其他列中的值累加起来。为了使这个示例尽量简单，我们仅仅使用了少量客户记录。但是，这种方法具有很好的扩展性，你可以在大量记录中使用这种方法执行计算，也可以修改一下代码，来处理多个输入文件中的数据。

至此，我们已经解决了为未知数目的类别计算统计量的问题。下面开始解决通过分析纯文本文件找出关键数据的问题。这个目标此刻听起来有点抽象，我们将在下一节对这个问题进行详细讨论，并给出解决方法。

5.3 为文本文件中数据的任意数目分类计算统计量

前面两个应用程序演示了如何通过 CSV 文件和 Excel 文件中的数据完成具体的任务。确实，本书的大部分内容都重点介绍如何分析和处理这些文件中的数据。对于 CSV 文件，我们使用 Python 内置的 `csv` 模块。对于 Excel 文件，我们下载并使用扩展模块 `xlrd`。CSV 文件和 Excel 文件是商业中常用的文件类型，所以知道如何处理这些文件是非常重要的。

同样，文本文件（也称平面文件）也是商业中常用的文件类型。前面已经说过，CSV 文件实际上就是以逗号分隔的文本文件形式保存的。活动日志、错误日志和交易记录是商业数据保存在文本文件中的几个更常见的例子。因为文本文件和 CSV 文件、Excel 文件一样，也经常应用于商业过程，但直到现在本书还没有详细介绍如何分析文本文件，所以我们使用下面这个应用程序演示如何从文本文件中提取数据并根据这些数据计算统计量。

正如前面一段提到的，错误日志通常是以文本文件形式存储的。MySQL 数据库系统中的错误日志就是以这种形式保存的。在上一章中，我们下载并安装了 MySQL 数据库系统，所以如果你在上一章中实现了示例程序，那么就可以访问 MySQL 错误日志了。但请记住，你可以根据自己的需要定位并访问 MySQL 系统的错误日志，完成本节中的示例并不要求你必须访问 MySQL 错误日志。

要在 Windows 系统中访问 MySQL 系统的错误日志文件，先打开资源管理器，选择 C 磁盘驱动器，然后打开 ProgramData，接着打开 MySQL 文件夹，再打开 MySQL Server <Version> 文件夹（例如，MySQL Server 5.6），最后打开 data 文件夹。在 data 文件夹中，应该有一个以扩展名 .err 结尾的错误日志文件。使用鼠标右击这个文件，用像 Notepad 或 Notepad++ 这样的文本编辑器打开这个文件，就可以看到 MySQL 系统的错误记录已经被写到这个日志文件中了。如果在这个路径下找不到文件夹，可以打开资源管理器，选择 C 磁盘驱动器，在右上角的搜索框中输入“.err”，然后等待系统找到错误日志文件。如果系统找到了多个错误日志文件，那么就选择与上面描述的路径最接近的那一个。

> macOS 用户应该可以在这里找到错误日志文件 /usr/local/mysql/data/<hostname>.err。

因为你的 MySQL 错误日志文件中的数据肯定与我的 MySQL 错误日志文件中的数据不一样，所以在这个应用程序中，我们使用一个独立的、有代表性的 MySQL 错误日志文件。这样就可以把重点放在 Python 代码上，而不是处理各种错误日志文件的不同之处。要为这个应用程序创建一个典型的 MySQL 错误日志文件，先打开一个文本编辑器，输入图 5-11 所示的各行文本，然后将文件保存为 mysql_server_error_log.txt。

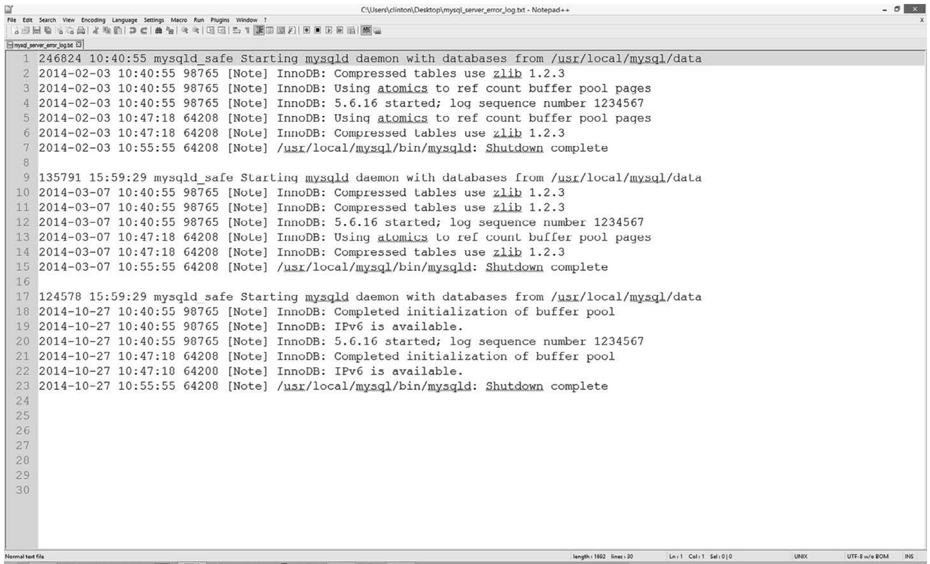


图 5-11：mysql_server_error_log.txt 中的 MySQL 数据库错误日志数据示例，显示在 Notepad++ 中

你可以看到，在 MySQL 错误日志文件中，记录着 mysqld 启动与结束的时间，以及在服务器运行时发生的各种关键错误的信息。例如，文件的第 1 行就显示了启动的时间，第 7 行则显示了这一天中 mysqld 结束的时间。第 2~6 行显示了这一天中当服务器运行时发生的关键错误。这些行的开头都是一个日期和时间戳，后面是由关键词 `[Note]` 引导的关键的错误消息。文件中其余各行包含着同样的信息，只是日期不同。

为了减少创建文件时的输入量，我重复了时间戳和很多关键错误消息。因此，创建文件时你只需输入 1~7 行，然后将这些行复制粘贴两次，修改一下日期和错误消息即可。

既然我们有了 MySQL 错误日志文件，那么下面就讨论一下商业应用。这种情况下的文本文件经常保存着一些分散的数据，可以进行分析、聚集和解释，以产生一些新的知识。举例来说，在我们这个应用中，错误日志文件记录着错误的类型和发生时间。在它的原始状态下，我们很难看出是否有某种错误发生的比其他错误更频繁，或者是否某种错误的频率会随着时间发生变化。通过解析这个文本文件，将相关信息聚集起来，然后以合适的形式写入一个输出文件，就可以从数据中获得知识，促使我们采取正确的行动。你使用的文本文件可能不是 MySQL 错误日志，但是，能够分析文本文件，找出关键数据，并将数据聚集起来以产生新知识，是处理文本文件的一项常用的重要技能。

既然我们已经理解了商业应用的需求，下面需要做的就是编写 Python 代码来进行错误消息的分析和计算。要完成这个操作，在文本编辑器中输入下列代码，然后将文件保存为 3parse_text_file.py：

```
#!/usr/bin/env python3
import sys
```

```
 3
 4 input_file = sys.argv[1]
 5 output_file = sys.argv[2]
 6 messages = { }
 7 notes = [ ]
 8 with open(input_file, 'r', newline='') as text_file:
 9     for row in text_file:
10         if '[Note]' in row:
11             row_list = row.split(' ', 4)
12             day = row_list[0].strip()
13             note = row_list[4].strip('\n').strip()
14             if note not in notes:
15                 notes.append(note)
16             if day not in messages:
17                 messages[day] = { }
18             if note not in messages[day]:
19                 messages[day][note] = 1
20             else:
21                 messages[day][note] += 1
22 filewriter = open(output_file, 'w', newline='')
23 header = ['Date']
24 header.extend(notes)
25 header = ','.join(map(str,header)) + '\n'
26 print(header)
27 filewriter.write(header)
28 for day, day_value in messages.items():
29     row_of_output = [ ]
30     row_of_output.append(day)
31     for index in range(len(notes)):
32          if notes[index] in day_value.keys():
33               row_of_output.append(day_value[notes[index]])
34          else:
35               row_of_output.append(0)
36     output = ','.join(map(str,row_of_output)) + '\n'
37     print(output)
38     filewriter.write(output)
39 filewriter.close()
```

在这个应用中，与 CSV 文件和 Excel 文件不同，因为我们分析的文本文件只包含纯文本，所以不需要导入 `csv` 或 `xlrd` 模块，只需要在第 2 和 3 行代码中导入 Python 内置的 `sys` 和 `string` 模块即可。前面已经介绍过，这两个模块分别可以使我们处理字符串和从命令行中读取输入参数。

第 4 和 5 行代码读取我们在命令行中提供的两个输入参数：包含 MySQL 错误日志的文本文件的路径名和 CSV 输出文件的路径名，CSV 输出文件中将保存每天发生的各种错误类型的信息。这两个输入参数被分别赋给两个变量：`input_file` 和 `output_file`。

第 6 行代码创建了一个空字典 `messages`。和前一个应用程序中的字典一样，`messages` 也是一个嵌套字典。外部字典的键是错误发生的具体日期，与之对应的值是另一个字典。内部字典的键是错误消息，与之对应的值是某一天错误发生的次数。

第 7 行代码创建了一个空列表 `notes`。列表 `notes` 将保存输入的错误日志文件中所有日期

发生的全部错误消息。将所有错误消息放在一个独立的数据结构中（也就是说放在一个列表中，而不是放在前面的字典中），可以更容易地在错误日志文件中搜索错误消息，将错误消息作为标题行写入输出文件，并在字典和列表中分别进行迭代，以将日期和数据计数写在输出文件中。

第 8 行代码使用 Python 的 with 语法打开输入文件以供读取。第 9 行代码创建了一个 for 循环，在输入文件的各行之间循环。

第 10 行代码是一个 if 语句，检验字符串 [Note] 是否在这一行中。包含字符串 [Note] 的行就是包含错误消息的行。你会发现没有与 if 语句相对应的 else 语句，所以代码对不包含字符串 [Note] 的行不做任何处理。对于包含字符串 [Note] 的行，第 11~21 行代码对这行数据进行解析，将某些特定的数据加载到前面的列表和字典中。

第 11 行代码使用 string 模块的 split() 方法按照空格将行进行拆分（最多使用 4 个空格拆分 4 次），然后将从每行中拆分出的 5 个部分赋给一个列表变量 row_list。我们限制了 split() 方法使用空格拆分的次数，因为前 4 个空格用来分隔出数据的 4 个不同片段，其余的空格出现在错误消息中，应该保留为错误消息的一部分。

第 12 行代码取出 row_list 中的第一个元素（一个日期），并除去日期两端任何多余的空格、制表符和换行符，然后将其赋给变量 day。

第 13 行代码取出 row_list 中的第五个元素（错误消息），并除去错误消息两端任何多余的空格、制表符和换行符，然后将其赋给变量 note。

第 14 行代码是一个 if 语句，检验变量 note 中的错误消息是否还没有包含在列表 notes 中。如果没有，那么第 15 行代码就使用 append() 方法将错误消息追加到列表中。通过检验错误消息是否存在，并仅将不在列表中的错误消息添加到列表中，就可以保证得到一个包含输入文件中所有不重复的错误消息的列表。

第 16 行代码是一个 if 语句，检验变量 day 中的日期是否还不是字典 messages 中的一个键。如果不是，那么第 17 行代码就将这个日期作为字典键添加到 messages 字典中，并创建一个空字典，作为与这个新的日期键对应的值。

第 18 行代码是一个 if 语句，检验变量 note 中的错误消息是否还不是内部字典中的一个键，这个内部字典是外部字典中与某个日期键对应的值。如果不是，那么第 19 行代码就将错误消息作为内部字典的键添加到内部字典中，并将与这个键对应的值设为 1，这个值是用来计数的。

第 20 行代码是一个 else 语句，是第 18 行中 if 语句的补充。这行代码处理一条具体错误消息在某一天出现多次的情况。当这种情况发生时，第 21 行代码将与这条错误消息对应的计数值增加 1。第 20 和 21 行代码保证与每条错误消息对应的计数值都能够反映出这条错误消息在某一天出现的次数。

当脚本处理了错误日志文件中的所有行之后，字典 messages 中将包含很多键-值对。这些键是有错误消息发生的所有不重复的日期，与其对应的值是另外一些字典，其中有各自的键-值对。这些内部字典中的键是发生于某个日期的不重复的错误消息，与这些键对应的值是一个整数计数值，表示这个错误消息在某个日期发生的次数。

第 22 行代码打开输出文件以供输出，并创建一个写文件对象 filewriter，用来将数据写入输出文件。

第 23 行代码创建了一个列表变量 header，并将字符串 Date 赋给这个列表。第 24 行代码使用 extend() 方法将列表变量 notes 中的内容扩展到列表变量 header 中。这样，header 中的第一个元素就是字符串 Date，其他元素则是来自于列表 notes 的不重复的错误消息。

第 25 行代码使用 str() 和 map() 函数以及 join() 方法，将列表变量 header 中的内容在写入输出文件之前转换成一个长字符串。map() 函数在 header 中的每个元素上应用 str() 函数，确保 header 中的每个元素都是字符串。然后使用 string 模块的 join() 方法在变量 header 中的每个字符串之间插入一个逗号，创建一个由逗号分隔各个值的长字符串。最后，向长字符串的最后添加一个换行符。这个长字符串包含由逗号分隔的列标题，末尾有一个换行符，将作为写入 CSV 输出文件中的第一行输出数据。

第 26 行代码在命令行窗口（或终端窗口）中打印出 header 中的值，也就是一个包含由逗号分隔的列标题的长字符串，这样我们就可以检查一下要写入输出文件的内容了。然后第 27 行代码使用 filewriter 对象的 write 方法将这个标题行写入输出文件。

第 28 行代码创建了一个 for 循环，并使用 items 函数在 messages 字典的键（变量 day）和值（变量 day_value）之间迭代。和前面的示例一样，第 29 行代码创建了一个空列表变量 row_of_output，用来保存写入输出文件的每行数据。因为我们已经将标题行写入了输出文件，所以知道第一列数据是日期。因此，应该将日期作为第一个追加到 row_of_output 中的数据。确实，第 30 行代码使用 append 方法将字典 messages 中的第一个日期追加到了 row_of_output 中。

此后，第 31~35 行代码在列表变量 notes 中的错误消息之间迭代，并判断在当前正在处理的日期中，每条错误消息是否发生。如果已经发生，代码就将与这条错误消息对应的计数值添加到行中的正确位置。如果在当前正在处理的日期中，这条错误消息没有发生，那么代码就将 0 添加到行中的正确位置。

特别地，第 31 行是一个 for 循环，使用 range 和 len 函数按照索引位置在 notes 中的各个值之间迭代。

第 32 行代码是一个 if 语句，检验 notes 中的每条错误消息是否出现在当前处理日期对应的错误消息列表中。换句话说，day_value 是与当前处理日期对应的内部字典，keys 函数创建了一个内部字典键的列表，内部字典的键就是在当前处理日期发生的错误消息。

对于 notes 中的每条错误消息，如果错误消息在当前处理日期发生，也就出现在了当前处理日期对应的错误消息列表中，那么第 33 行代码就使用 append 方法将与错误消息对应的计数值追加到 row_of_output 中。通过使用 notes 中错误消息的索引值，可以保证为每条错误消息提取出正确的计数值。

例如，列表 notes 中的第一条错误消息是 InnoDB: Compressed tables use zlib 1.2.3，你可以使用 notes[0] 引用这个字符串。当你运行这个脚本时，在命令行窗口或终端窗口中可以看到，这个字符串是输出文件中第二列的标题。从屏幕上的输出还可以看出，这条错误消息在 2014-03-07 出现了 3 次。

再回顾一下脚本中的第 28~35 行代码，并记住这个日期和错误消息，看看代码是如何将正确的数据写到输出文件中的。第 28 行代码创建了一个 for 循环，在字典 messages 中的所有日期之间循环，所以在某个时候，for 循环会处理到 2014-03-07 这个日期。当第 28 行代码开始处理 2014-03-07 时，代码已经准备好了内部字典中的错误消息和与之对应的计数值（因为它们就是 day_value 字典中的键和值）。在处理 2014-03-07 时，第 31 行代码创建了另一个 for 循环，在列表 notes 的索引值之间循环。第一次循环时，索引值为 0，所以在第 32 行代码中，notes[0] 等于 InnoDB: Compressed tables use zlib 1.2.3。因为正在处理 2014-03-07 这个日期，notes[0] 中的值位于与这个日期对应的内部字典的键的集合中，所以第 32 行代码为 True，第 33 行代码被执行。在第 33 行代码中，notes[index] 就是 notes[0]，day_value[notes[0]] 就是 day_value["InnoDB: Compressed tables use zlib 1.2.3"]，这个表达式指向的就是内部字典中与这个键对应的值，对于 2014-03-07，这个值就是整数 3。以上操作的结果就是，在输出文件的 2014-03-07 这一行中，错误消息 InnoDB: Compressed tables use zlib 1.2.3 这一列的值为 3。

第 34 和 35 行代码处理列表 notes 中的错误消息没有出现在一个特定日期的错误消息列表中的情况。在这种情况下，第 35 行代码向输出行中正确的列的位置添加一个 0。例如，列表 notes 中的最后一条错误消息是 InnoDB:IPv6 is available.。这条错误消息没有发生在 2014-03-07，所以在 2014-03-07 这条输出行中，需要在对应这条错误消息的列中记录一个 0。当第 32 行代码检验 notes 中的最后一个值（notes[5]，InnoDB:IPv6 is available.）是否在内部字典的键集合中时，结果就是 Fasle，所以执行第 34 行代码中的 else 语句，使用第 35 行代码将 0 添加到 row_of_output 的最后一列中。其余日期和错误消息的计数值也是以同样的方式添加的。

第 36 行代码使用与第 25 行代码同样的处理过程，将列表变量 row_of_output 中的内容在写入输出文件之前转换成一个长字符串。map 函数对列表变量 row_of_output 中的每个元素使用 str 函数，保证变量中的每个元素都是一个字符串。然后使用 string 模块的 join 方法在变量 row_of_output 中的每个字符串之间插入一个逗号，创建一个由逗号分隔各个值的长字符串。最后，向长字符串的最后添加一个换行符。这个长字符串包含由逗号分隔的列标题，末尾有一个换行符，被赋给变量 output，将作为一行输出写入 CSV 输出文件。

第 37 行代码在命令行窗口或终端窗口打印出 output 中的值，也就是一个由逗号分隔各个值的长字符串，这样你可以检查一下要写入输出文件的内容。然后第 38 行代码使用 filewriter 对象的 write 方法将这行输出写入输出文件。

最后，第 39 行代码使用 filewriter 的 close 方法关闭 filewriter 对象。

我们已经完成了 Python 脚本，现在可以使用脚本计算不同的错误随着时间变化发生的次数，并将结果写入 CSV 输出文件了。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
python 3parse_text_file.py mysql_server_error_log.txt\
output_files\3app_output.csv
```

你可以看到在命令行窗口或终端窗口打印出的输出，如图 5-12 所示。

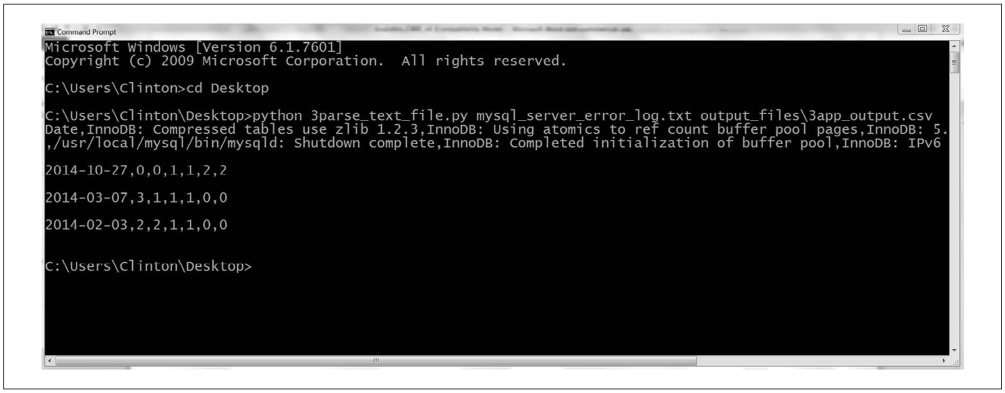


图 5-12：在 MySQL 错误日志文件 mysql_server_error_log.txt 上运行 3parse_text_file.py 的结果，显示在命令行窗口中

打印在命令行窗口中的输出展示了写在输出文件 3app_output.csv 中的数据。输出的第一行是标题行，展示了输出文件中所有列的标题。第一个列标题 Date 说明第一列的内容是输入文件中记录错误消息的日期。这一列说明输入文件中包含 3 个不重复的日期。其余 6 个列标题是输入文件中出现的错误消息，因此输入文件中包含 6 个不重复的错误消息。这 6 列中包含了输入文件中一个具体的错误消息在每个日期出现的次数。例如，输出中的最后一列表示错误消息 `InnoDB:IPv6 is available.` 在 2014-10-27 出现了 2 次，在 2014-02-03 出现了 0 次，在 2014-03-07 出现了 0 次。

输出文件 3app_output.csv 中的内容和打印在命令行窗口中的输出一样，如图 5-13 所示。

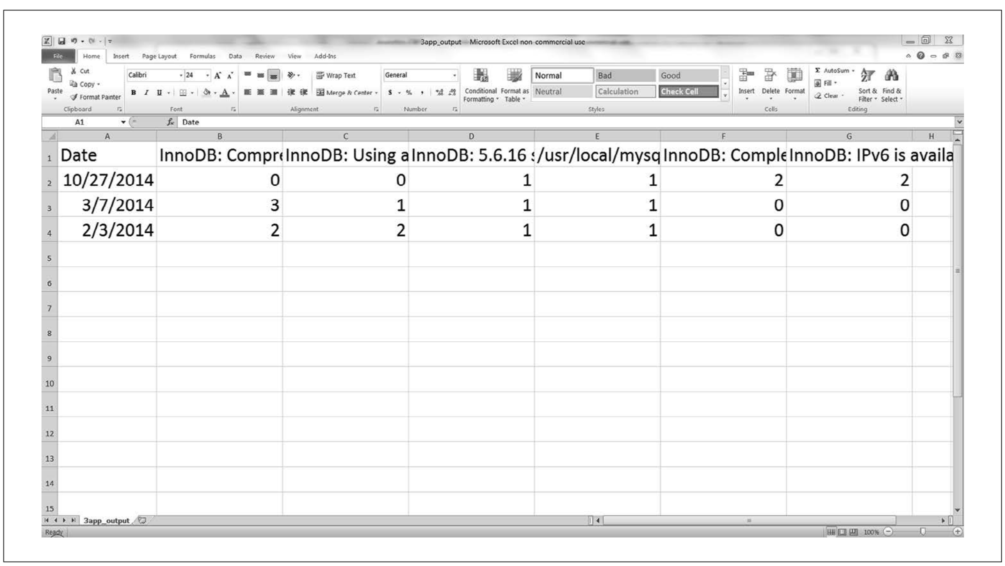


图 5-13：CSV 文件 3app_output.csv 中的 3parse_tex_file.py 的输出（某个错误消息在某一天发生的次数），显示在 Excel 工作表中

这个屏幕截图是在 Excel 中打开的 CSV 输出文件，显示了写在输出文件中的数据。在标题行中，日期标题后面是 6 个错误消息，由于横向空间的限制，在屏幕截图中你看不到完整的错误消息（如果你创建了这个文件，可以通过增加列的宽度，看看完整的消息）。

你可以看到，在输出文件中，行表示某个日期，列表示某个错误消息，这使得这个文件看上去比较怪。在这个简单的例子中，错误消息（列）的数量多于日期（行）的数量，似乎更应该使用行来表示错误消息。通常，在更大的日志文件中，日期的数量很可能比错误消息多。在这种情况下，就更应该用行表示日期（因为行更多一些），用列表示错误消息。最终，是用行表示日期，列表示错误消息，还是相反，取决于你的实际分析需要和个人偏好。你可以修改现有代码将输出进行转置，用列表示日期，用行表示错误消息，这会是一个很好的练习。

这个应用程序综合运用我们在第 1 章中学习的几种技术（例如填充一个嵌套字典）来解决一个常见的实际问题。商业分析师经常需要分析文本文件，找出关键数据，并将数据进行聚集或摘要，以获取新的知识。在很多情况下，我们需要以不同方式处理成千上万条数据，所以手动分析数据是不可能的。

这一节演示了一种可扩展的方法，这种方法可以从文本文件中解析出数据，并使用解析出的数据计算基本的统计量。为了使这个示例尽量简单，我们仅仅使用了一个很短的错误日志文件。但是，这种方法具有很好的扩展性，你可以使用它来解析更大的日志文件，也可以修改一下代码，来处理多个文本文件中的数据。

5.4 本章练习

(1) 第一个应用程序在保存于一个文件夹中的输入文件中进行搜索。但是，有些时候输入文件是保存在几个嵌套的文件夹中的。修改第一个应用程序中的代码，使脚本可以遍历一组嵌套文件夹并处理保存在所有文件夹中的输入文件。提示：在 Internet 上搜索“Python os walk”[2]。

(2) 修改第二个应用程序，计算你从铜牌服务包、银牌服务包和金牌服务包客户中获得的收入，假设铜牌服务包收入为 \$20/ 月，银牌服务包收入为 \$40/ 月，金牌服务包收入为 \$50/ 月。

(3) 练习使用字典来为数据分类。例如，将 Excel 工作表或 CSV 文件中的数据解析成一个字典，使输入文件中每一列都是字典中的一个键–值对。也就是说，每列标题是字典中的键，每个键对应的值就是相应列中的数据。

注 2：你可以在 https://docs.python.org/3/library/os.html 和 http://www.pythoncentral.io/how-to-traverse-a-directory-tree-in-python-guide-to-os-walk/ 上获得关于遍历目录树的更多信息。

第6章

图与图表

创建图与图表是很多分析项目中的一个重要步骤，它通常是项目开始时探索性数据分析（EDA）的一部分，或者在项目报告阶段向其他人介绍你的数据分析结果时使用。数据可视化可以使你看到变量的分布和变量之间的关系，还可以检查建模过程中的假设。

Python 提供了若干种用于绘图的扩展包，包括 `matplotlib`、`pandas`、`ggplot` 和 `seaborn`。因为 `matplotlib` 是最基础的扩展包，它为 `pandas` 和 `seaborn` 提供了一些基础的绘图概念和语法，所以这里先介绍 `matplotlib`。然后，我们通过一些示例，看看其他扩展包如何提供更简洁的绘图语法，或者提供一些额外的功能。

6.1 matplotlib

`matplotlib` 是一个绘图库，创建的图形可达到出版的质量要求。它可以创建常用的统计图，包括条形图、箱线图、折线图、散点图和直方图。它还有一些扩展工具箱，比如 `basemap` 和 `cartopy`，用于制作地图，以及 `mplot3d`，用于进行 3D 绘图。

`matplotlib` 提供了对图形各个部分进行定制的功能。例如，它可以设置图形的形状和大小、x 轴与 y 轴的范围和标度、x 轴与 y 轴的刻度线和标签、图例以及图形的标题。你可以参考一下 `matplotlib` 初学者指南和 API（http://matplotlib.org/users/beginner.html），以获得更多的关于定制图形的信息。

下面的示例演示了如何使用 `matplotlib` 创建最常用的统计图形。

6.1.1 条形图

条形图表示一组分类数值，比如计数值。常用的条形图包括垂直图、水平图、堆积图和分组图。下面的脚本 bar_plot.py 演示了如果创建一个垂直条形图：

```
 1 #!/usr/bin/env python3
 2 import matplotlib.pyplot as plt
 3 plt.style.use('ggplot')
 4 customers = ['ABC', 'DEF', 'GHI', 'JKL', 'MNO']
 5 customers_index = range(len(customers))
 6 sale_amounts = [127, 90, 201, 111, 232]
 7 fig = plt.figure()
 8 ax1 = fig.add_subplot(1,1,1)
 9 ax1.bar(customers_index, sale_amounts, align='center', color='darkblue')
10 ax1.xaxis.set_ticks_position('bottom')
11 ax1.yaxis.set_ticks_position('left')
12 plt.xticks(customers_index, customers, rotation=0, fontsize='small')
13 plt.xlabel('Customer Name')
14 plt.ylabel('Sale Amount')
15 plt.title('Sale Amount per Customer')
16 plt.savefig('bar_plot.png', dpi=400, bbox_inches='tight')
17 plt.show()
```

第 2 行代码是惯常的 `import` 语句。第 3 行代码使用 `ggplot` 样式表来模拟 `ggplot2` 风格的图形，`ggplot2` 是一个常用的 R 语言绘图包。

第 4、5 和 6 行代码为条形图准备数据。我创建了一个客户索引列表，因为 `xticks` 函数在设置标签时要求索引位置和标签值。

使用 `matplotlib` 绘图时，首先要创建一个基础图，然后在基础图中创建一个或多个子图。脚本中的第 7 行代码创建了一个基础图。第 8 行代码向基础图中添加了一个子图。因为可以向基础图中添加多个子图，所以必须指定要创建几行和几列子图，以及使用哪个子图。`1，1，1` 表示创建 1 行 1 列的子图，并使用第 1 个也是唯一的一个子图。

第 9 行代码创建条形图。`customer_index` 设置条形左侧在 x 轴上的坐标。`sale_amounts` 设置条形的高度。`align='center'` 设置条形与标签中间对齐。`color='darkblue'` 设置条形的颜色。

第 10 和 11 行代码通过设置刻度线位置在 x 轴底部和 y 轴左侧，使图形的上部和右侧不显示刻度线。

第 12 行代码将条形的刻度线标签由客户索引值更改为实际的客户名称。`rotation=0` 表示刻度标签应该是水平的，而不是倾斜一个角度。`fontsize='small'` 将刻度标签的字体设为小字体。

第 13、14 和 15 行代码向图中添加 x 轴标签、y 轴标签和图形标题。

第 16 行代码将统计图保存在当前文件夹中，文件名为 bar_plot.png。`dpi=400` 设置图形分辨率 [每英寸（1 英寸 =2.54 厘米）的点数）]，`bbox_inches='tight'` 表示在保存图形时，将图形四周的空白部分去掉。

第 17 行代码指示 `matplotlib` 在一个新窗口中显示统计图。结果如图 6-1 所示。

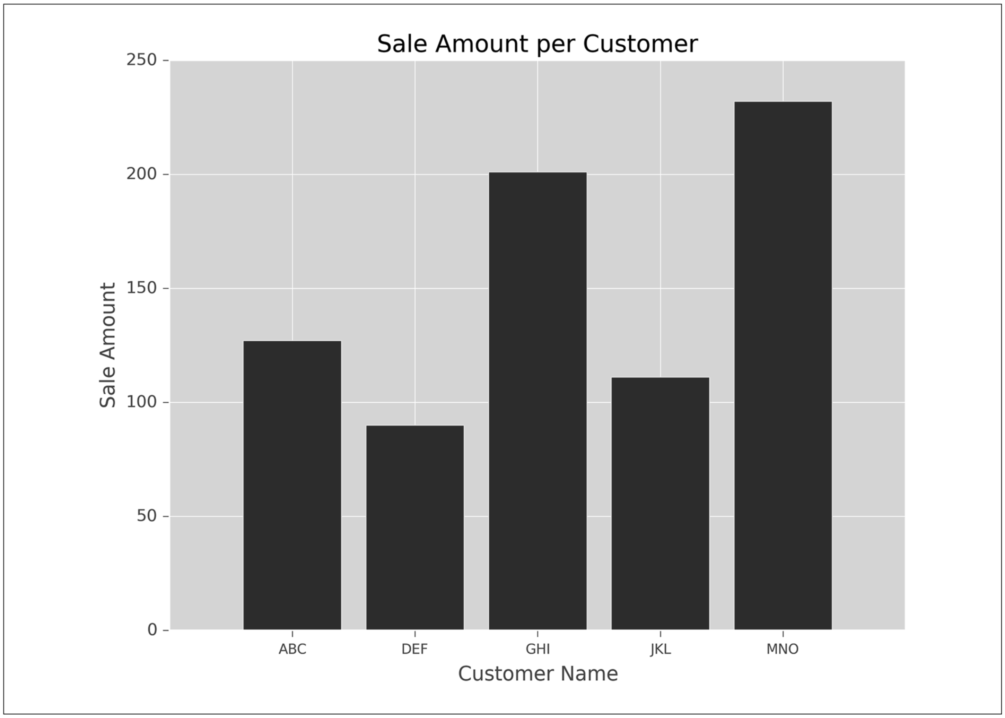


图 6-1：`matplotlib` 制作的条形图

6.1.2　直方图

直方图用来表示数值分布。常用的直方图包括频率分布、频率密度分布、概率分布和概率密度分布。下面的脚本 histogram.py 演示了如何创建一个频率分布图：

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
mu1, mu2, sigma = 100, 130, 15
x1 = mu1 + sigma*np.random.randn(10000)
x2 = mu2 + sigma*np.random.randn(10000)
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
n, bins, patches = ax1.hist(x1, bins=50, normed=False, color='darkgreen')
n, bins, patches = ax1.hist(x2, bins=50, normed=False, color='orange', alpha=0.5)
ax1.xaxis.set_ticks_position('bottom')
ax1.yaxis.set_ticks_position('left')
plt.xlabel('Bins')
plt.ylabel('Number of Values in Bin')
fig.suptitle('Histograms', fontsize=14, fontweight='bold')
ax1.set_title('Two Frequency Distributions')
plt.savefig('histogram.png', dpi=400, bbox_inches='tight')
plt.show()
```

第 6 和 7 行代码使用 Python 的随机数生成器创建两个正态分布变量 `x1` 和 `x2`。`x1` 的均值是 100，`x2` 的均值是 130，所以两个分布会有一些重叠，但不会是一个覆盖掉另一个。第 10 和 11 行代码为这两个变量创建两个柱形图，或称频率分布图。`bins=50` 表示每个变量的值应该被分成 50 份。`normed=False` 表示直方图显示的是频率分布，而不是概率密度。第一个直方图是暗绿色，第二个直方图是橙色。`alpha=0.5` 表示第二个直方图应该是透明的，这样我们就可以看到两个直方图重叠部分的暗绿色图。

第 16 行代码为基础图添加一个居中的标题，字体大小为 14，粗体。第 17 行代码为子图添加一个居中的标题，位于基础图标题下面。我们用这两行代码为统计图创建标题和副标题，如图 6-2 所示。

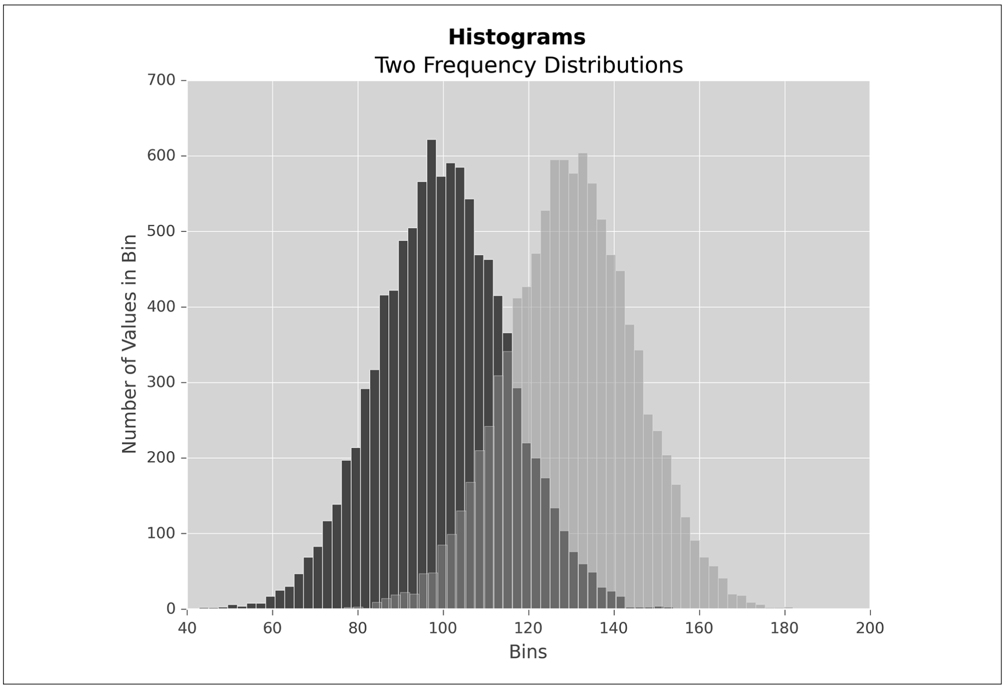


图 6-2：`matplotlib` 制作的两个直方图

6.1.3 折线图

折线图中的数值点在一条折线上。它通常用来表示数据随着时间发生的变化。下面的脚本 line_plot.py 演示了如何创建一幅折线图：

```
#!/usr/bin/env python3
from numpy.random import randn
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plot_data1 = randn(50).cumsum()
plot_data2 = randn(50).cumsum()
plot_data3 = randn(50).cumsum()
```

```
 8 plot_data4 = randn(50).cumsum()
 9 fig = plt.figure()
10 ax1 = fig.add_subplot(1,1,1)
11 ax1.plot(plot_data1, marker=r'o', color=u'blue', linestyle='-',\
12 label='Blue Solid')
13 ax1.plot(plot_data2, marker=r'+', color=u'red', linestyle='--',\
14 label='Red Dashed')
15 ax1.plot(plot_data3, marker=r'*', color=u'green', linestyle='-.',\
16 label='Green Dash Dot')
17 ax1.plot(plot_data4, marker=r's', color=u'orange', linestyle=':',\
18 label='Orange Dotted')
19 ax1.xaxis.set_ticks_position('bottom')
20 ax1.yaxis.set_ticks_position('left')
21 ax1.set_title('Line Plots: Markers, Colors, and Linestyles')
22 plt.xlabel('Draw')
23 plt.ylabel('Random Number')
24 plt.legend(loc='best')
25 plt.savefig('line_plot.png', dpi=400, bbox_inches='tight')
26 plt.show()
```

同样，我们在第 5~8 行代码中使用 `randn` 创建绘图所用的随机数据。第 11~18 行代码创建 4 条折线。每条折线都可以通过选项进行设置，使用不同的数据点类型、颜色和线型。`label` 参数保证折线在图例中可以正确标记。

第 24 行代码为统计图创建图例。`loc='best'` 指示 `matplotlib` 根据图中的空白部分将图例放在最合适的位置。或者，你也可以使用这个参数为图例指定一个具体位置。图 6-3 展示了这个脚本创建的折线图。

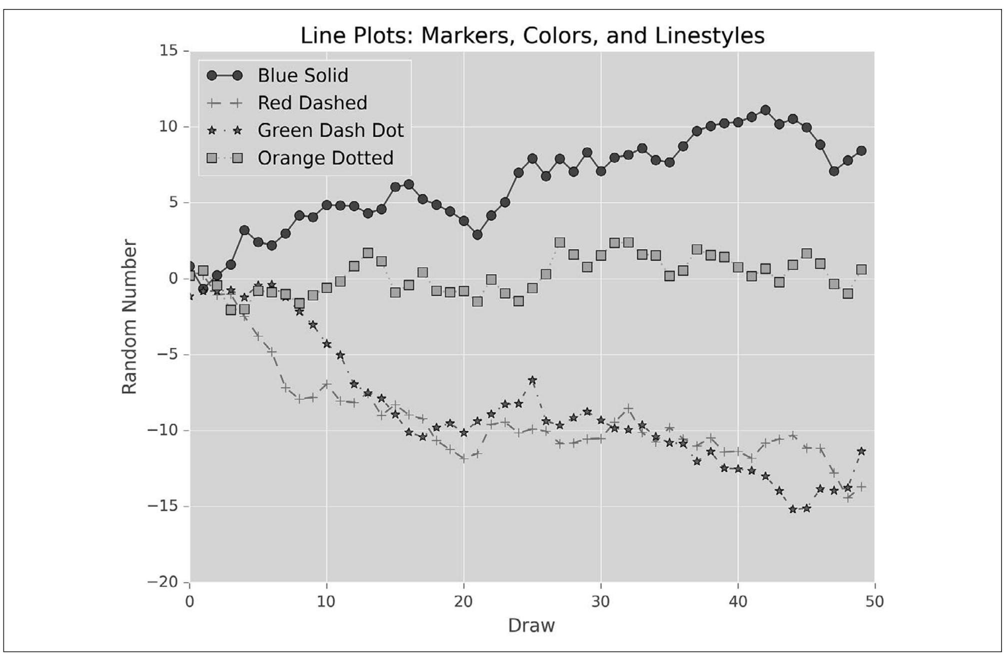


图 6-3：`matplotlib` 创建的折线图（包含 4 条折线）

6.1.4 散点图

散点图表示两个数值变量之间的相对关系，这两个变量分别位于两个数轴上。例如，身高与体重，或者供给与需求。散点图有助于识别出变量之间是否具有正相关（图中的点集中于某个具体参数）或负相关（图中的点像云一样发散）。你还可以画一条回归曲线，也就是使方差最小的曲线，通过图中的点基于一个变量的值预测另一个变量的值。

下面的脚本 scatter_plot.py 演示了如何创建一幅散点图，并在各个点之间画一条回归曲线：

```
 1 #!/usr/bin/env python3
 2 import numpy as np
 3 import matplotlib.pyplot as plt
 4 plt.style.use('ggplot')
 5 x = np.arange(start=1., stop=15., step=1.)
 6 y_linear = x + 5. * np.random.randn(14.)
 7 y_quadratic = x**2 + 10. * np.random.randn(14.)
 8 fn_linear = np.poly1d(np.polyfit(x, y_linear, deg=1))
 9 fn_quadratic = np.poly1d(np.polyfit(x, y_quadratic, deg=2))
10 fig = plt.figure()
11 ax1 = fig.add_subplot(1,1,1)
12 ax1.plot(x, y_linear, 'bo', x, y_quadratic, 'go', \
13               x, fn_linear(x), 'b-', x, fn_quadratic(x), 'g-', linewidth=2.)
14 ax1.xaxis.set_ticks_position('bottom')
15 ax1.yaxis.set_ticks_position('left')
16 ax1.set_title('Scatter Plots Regression Lines')
17 plt.xlabel('x')
18 plt.ylabel('f(x)')
19 plt.xlim((min(x)-1., max(x)+1.))
20 plt.ylim((min(y_quadratic)-10., max(y_quadratic)+10.))
21 plt.savefig('scatter_plot.png', dpi=400, bbox_inches='tight')
22 plt.show()
```

这里我使用了一点小技巧，在第 6 和 7 行代码中，通过随机数使数据与一条直线和一条二次曲线稍稍偏离。然后，在第 8 和 9 行代码中，使用 `numpy` 的 `polyfit` 函数通过两组数据点 `(x, y_linear)` 和 `(x, y_quadratic)` 拟合出一条直线和一条二次曲线，再使用 `poly1d` 函数根据直线和二次曲线的参数生成一个线形方程和二次方程。如果是实际数据，你可以使用 `polyfit` 函数计算出某个阶数的多项式拟合模型的系数。`poly1d` 函数可以使用这些系数创建实际的多项式方程。

第 12 行代码创建带有两条回归曲线的散点图。`'bo'` 表示 `(x, y_linear)` 点是蓝色圆圈，`'go'` 表示 `(x, y_quadratic)` 点是绿色圆圈。同样，`'b-'` 表示 `(x, y_linear)` 点之间的线是一条蓝色实线，`'g-'` 表示 `(x, y_quadratic)` 点之间的线是一条绿色实线。通过 `linewidth` 可以设置线的宽度。

你可以尝试改变这些显示变量，做出符合自己风格的统计图。

第 19 和 20 行代码设置了 x 轴和 y 轴的范围。这两条曲线使用 `min` 和 `max` 函数基于实际数据设置坐标轴范围。你也可以使用具体的数值设置范围，例如 `xlim(0, 20)` 和 `ylim(0, 200)`。如果你没有设置坐标轴范围，那么 `matplotlib` 会替你设置。脚本运行的结果如图 6-4 所示。

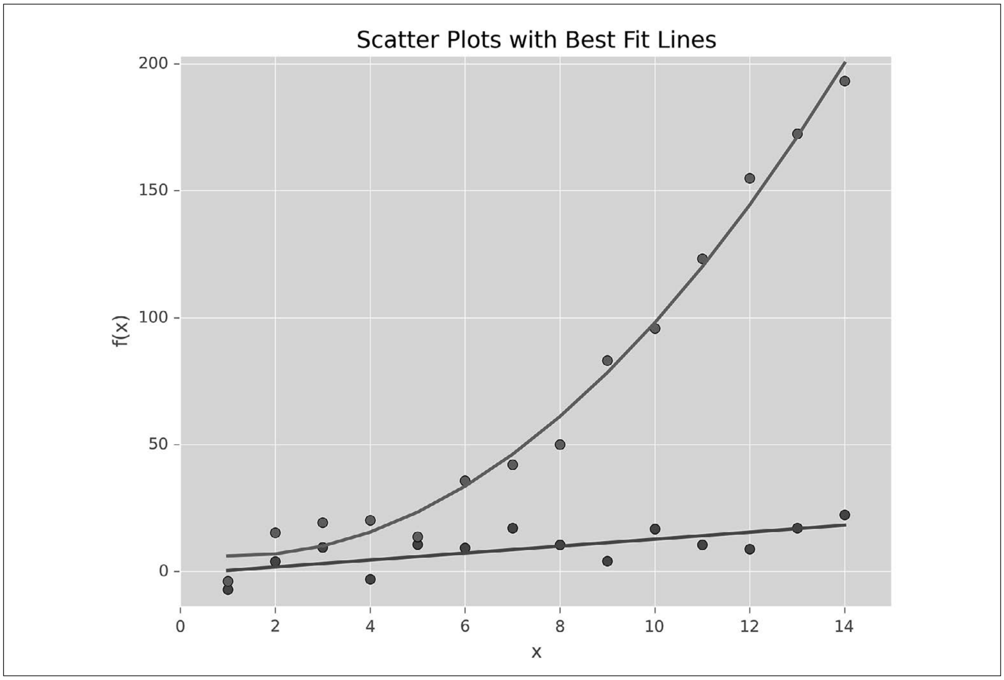


图 6-4：matplotlib 生成的散点图以及线形拟合曲线和二次拟合曲线

6.1.5　箱线图

箱线图可以表示出数据的最小值、第一四分位数、中位数、第三四分位数和最大值。箱体的下部和上部边缘线分别表示第一四分位数和第三四分位数，箱体中间的直线表示中位数。箱体上下两端延伸出去的直线（whisker，亦称为“须”）表示非离群点的最小值和最大值，在直线（须）之外的点表示离群点。

下面的脚本 box_plot.py 演示了如何创建一幅箱线图：

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
N = 500
normal = np.random.normal(loc=0.0, scale=1.0, size=N)
lognormal = np.random.lognormal(mean=0.0, sigma=1.0, size=N)
index_value = np.random.random_integers(low=0, high=N-1, size=N)
normal_sample = normal[index_value]
lognormal_sample = lognormal[index_value]
box_plot_data = [normal,normal_sample,lognormal,lognormal_sample]
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
box_labels = ['normal','normal_sample','lognormal','lognormal_sample']
ax1.boxplot(box_plot_data, notch=False, sym='.', vert=True, whis=1.5, \
                    showmeans=True, labels=box_labels)
```

```
17 ax1.xaxis.set_ticks_position('bottom')
18 ax1.yaxis.set_ticks_position('left')
19 ax1.set_title('Box Plots: Resampling of Two Distributions')
20 ax1.set_xlabel('Distribution')
21 ax1.set_ylabel('Value')
22 plt.savefig('box_plot.png', dpi=400, bbox_inches='tight')
23 plt.show()
```

第 14 行代码创建了一个列表 `box_labels`，里面保存着每个箱线图的标签。在下一行代码中的 `boxplot` 函数中将使用这个列表。

第 15 行代码使用 `boxplot` 函数创建 4 个箱线图。`notch=False` 表示箱体是矩形，而不是在中间收缩。`sym='.'` 表示离群点（就是直线之外的点）使用圆点，而不是默认的 + 符号。`vert=True` 表示箱体是垂直的，不是水平的。`whis=1.5` 设定了直线从第一四分位数和第三四分位数延伸出的范围（例如：Q3+whis*IQR，IQR 就是四分位距，等于 Q3-Q1）。`showmeans=True` 表示箱体在显示中位数的同时也显示均值。`labels=box_labels` 表示使用 `box_labels` 中的值来标记箱线图。

这个脚本的运行结果如图 6-5 所示。

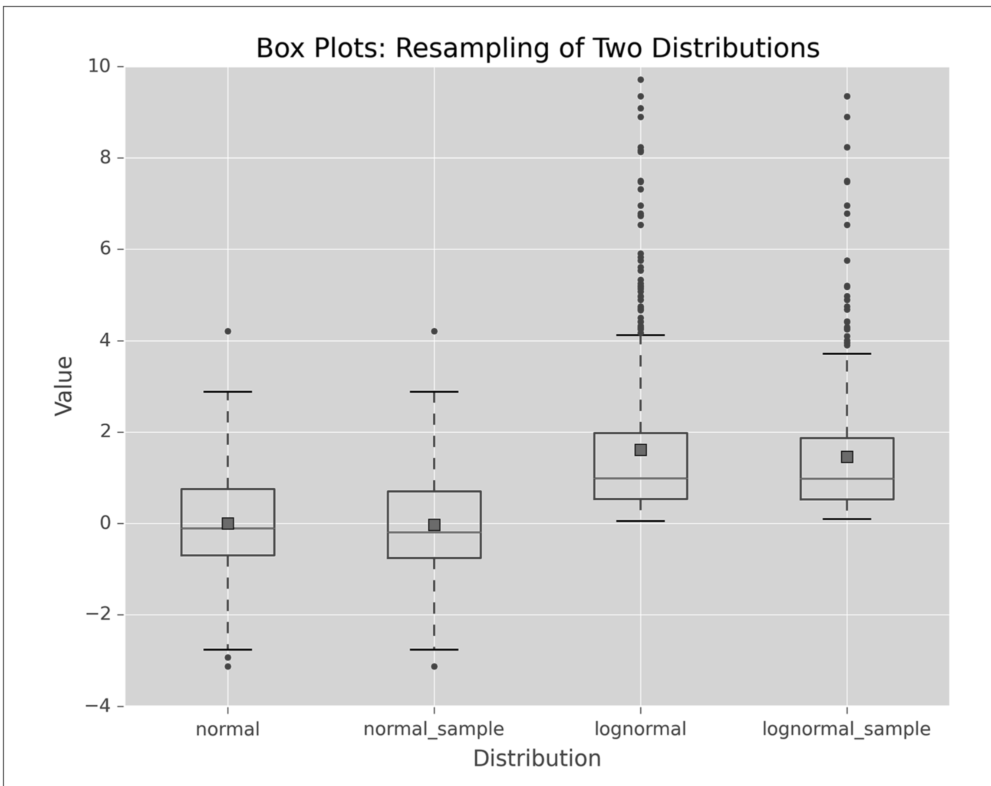


图 6-5：正态分布和对数正态分布数据的箱线图，以及这两个分布中抽样数据的箱线图，由 `matplotlib` 生成

6.2 pandas

pandas 通过提供一个可以作用于序列和数据框的函数 plot，简化了基于序列和数据框中的数据创建图表的过程。plot 函数默认创建折线图，你还可以通过设置参数 kind 创建其他类型的图表。

例如，除了使用 matplotlib 创建标准统计图，还可以使用 pandas 创建其他类型的统计图，比如六边箱图（hexagonal bin plot）、矩阵散点图、密度图、Andrews 曲线图、平行坐标图、延迟图、自相关图和自助抽样图。如果要向统计图中添加第二 y 轴、误差棒和数据表，使用 pandas 可以很直接地实现。

为了说明在 pandas 中创建统计图的方法，我们通过下面的脚本 pandas_plots.py 来演示如何使用数据框中的数据创建一对条形图和箱线图，并将它们并排放置：

```
 1 #!/usr/bin/env python3
 2 import pandas as pd
 3 import numpy as np
 4 import matplotlib.pyplot as plt
 5 plt.style.use('ggplot')
 6 fig, axes = plt.subplots(nrows=1, ncols=2)
 7 ax1, ax2 = axes.ravel()
 8 data_frame = pd.DataFrame(np.random.rand(5, 3),
 9     index=['Customer 1', 'Customer 2', 'Customer 3', 'Customer 4', 'Customer 5'],
10     columns=pd.Index(['Metric 1', 'Metric 2', 'Metric 3'], name='Metrics'))
11 data_frame.plot(kind='bar', ax=ax1, alpha=0.75, title='Bar Plot')
12 plt.setp(ax1.get_xticklabels(), rotation=45, fontsize=10)
13 plt.setp(ax1.get_yticklabels(), rotation=0, fontsize=10)
14 ax1.set_xlabel('Customer')
15 ax1.set_ylabel('Value')
16 ax1.xaxis.set_ticks_position('bottom')
17 ax1.yaxis.set_ticks_position('left')
18 colors = dict(boxes='DarkBlue', whiskers='Gray', medians='Red', caps='Black')
19 data_frame.plot(kind='box', color=colors, sym='r.', ax=ax2, title='Box Plot')
20 plt.setp(ax2.get_xticklabels(), rotation=45, fontsize=10)
21 plt.setp(ax2.get_yticklabels(), rotation=0, fontsize=10)
22 ax2.set_xlabel('Metric')
23 ax2.set_ylabel('Value')
24 ax2.xaxis.set_ticks_position('bottom')
25 ax2.yaxis.set_ticks_position('left')
26 plt.savefig('pandas_plots.png', dpi=400, bbox_inches='tight')
27 plt.show()
```

第 6 行代码创建了一个基础图和两个并排放置的子图。第 7 行代码使用 ravel 函数将两个子图分别赋给两个变量 ax1 和 ax2，这样我们就不必使用行和列的索引（例如，axes[0,0] 和 axes[0,1]）来引用子图了。

第 11 行代码使用 pandas 的 plot 函数在左侧子图中创建了一个条形图。第 12 和 13 行代码使用 matplotlib 的函数来设置 x 轴和 y 轴标签的旋转角度和字体大小。

第 18 行代码为箱线图单独创建了一个颜色字典。第 19 行代码在右侧子图中创建了一个箱线图，使用 colors 变量为箱线图各部分着色，并将离群点的形状设置为红色圆点。

脚本生成的条形图和箱线图如图 6-6 所示。你可以在 pandas 绘图文档（http://Pandas.pydata.org/Pandas-docs/stable/visualization.html）中学习到创建和设置统计图表的更多信息。

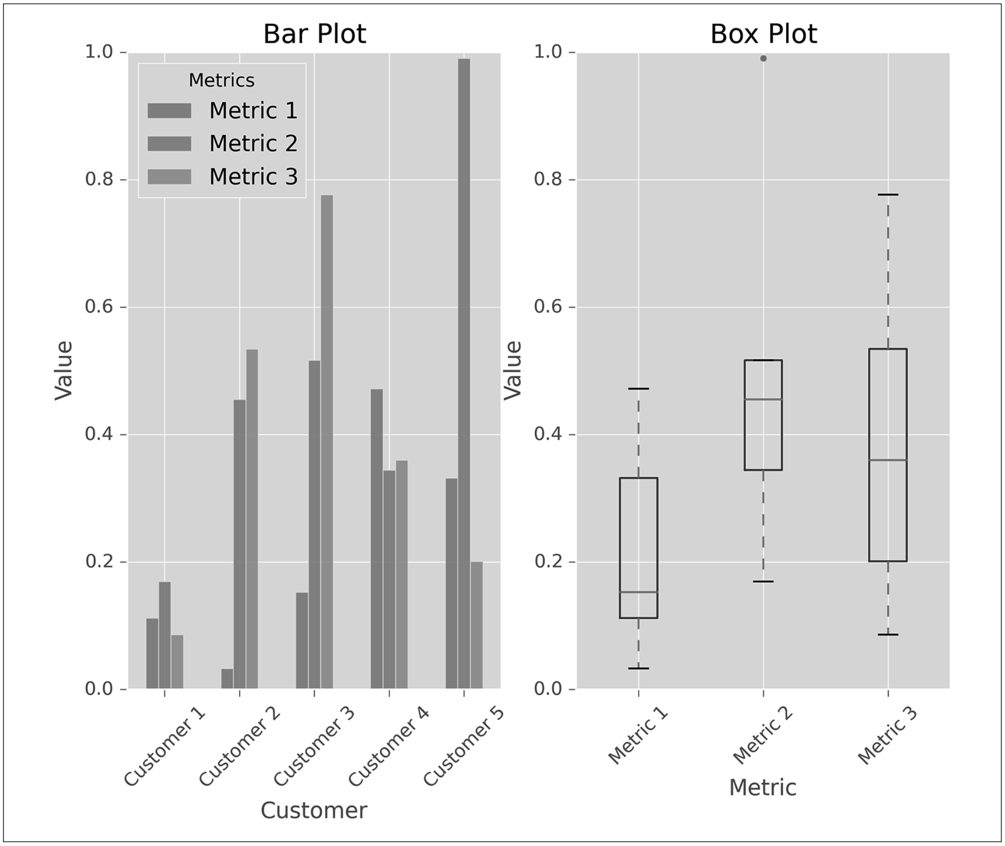


图 6-6：pandas 生成的并排放置的条形图和箱线图

6.3 ggplot

ggplot 是一个 Python 绘图包，它基于 R 语言的 ggplot2 包和图形语法。ggplot 与其他绘图包的关键区别是它的语法将数据与实际绘图明确地分离开来。为了对数据进行可视化表示，ggplot 提供了几种基本元素：几何对象、图形属性和标度。除此之外，为了进行更高级的绘图，ggplot 还提供一些附加元素：统计变换、坐标系、子窗口和可视化主题。要想获得关于 ggplot 的详细信息，可以参考 Hadley Wickham 的著作《ggplot2：数据分析与图形艺术（第 2 版）》，也可以参考 Leland Wilkinson 的著作：*Grammar of Graphics, 2nd Edition*（Springer）。

Python 的 ggplot 库不像 R 语言的 ggplot2 库那样成熟，所以它不具备 ggplot2 的所有功能，也就是说，它没有那么多的几何对象、统计变换和标度，也没有坐标系、注释和增强功能。在与 ggplot 相关的包进行了升级与修改之后，使用 ggplot 也可能会遇到问题。例

如，有一次我使用 ggplot 创建直方图时就遇到了问题，因为 pandas 中 pivot_table 方法的关键字 rows 和 cols 被去掉了，替换成了 index 和 columns。在网上搜索了解决方案之后，我发现应该将脚本 ggplot/stats/stat_bin.py 的某行代码中的“rows”修改为“index”，才能解决这个问题。

和本章中介绍的其他 Python 绘图包相比，ggplot 的缺点很明显，因此我推荐使用其他绘图包来创建统计图。但是，因为我是 R 语言 ggplot2 包的疯狂粉丝，所以在本节中我要介绍一下 ggplot。如果你精通 R 语言并且熟悉 ggplot2，那么马上就可以使用 ggplot 创建统计图，只要你的需求在它的功能范围内。

下面的脚本 ggplot_plots.py 演示了如何通过 ggplot 创建一些基础统计图，使用的数据就是 ggplot 包内部的数据：

```
#!/usr/bin/env python3
from ggplot import *
print(mtcars.head())
plt1 = ggplot(aes(x='mpg'), data=mtcars) +\
        geom_histogram(fill='darkblue', binwidth=2) +\
        xlim(10, 35) + ylim(0, 10) +\
        xlab("MPG") + ylab("Frequency") +\
        ggtitle("Histogram of MPG") +\
        theme_matplotlib()
print(plt1)
print(meat.head())
plt2 = ggplot(aes(x='date', y='beef'), data=meat) +\
        geom_line(color='purple', size=1.5, alpha=0.75) +\
        stat_smooth(colour='blue', size=2.0, span=0.15) +\
        xlab("Year") + ylab("Head of Cattle Slaughtered") +\
        ggtitle("Beef Consumption Over Time") +\
        theme_seaborn()
print(plt2)
print(diamonds.head())
plt3 = ggplot(diamonds, aes(x='carat', y='price', colour='cut')) +\
        geom_point(alpha=0.5) +\
        scale_color_gradient(low='#05D9F6', high='#5011D1') +\
        xlim(0, 6) + ylim(0, 20000) +\
        xlab("Carat") + ylab("Price") +\
        ggtitle("Diamond Price by Carat and Cut") +\
        theme_gray()
print(plt3)
ggsave(plt3, "ggplot_plots.png")
```

这 3 个统计图是使用 ggplot 包中的 3 个数据集 mtcars、meat 和 diamonds 创建的。在创建统计图之前，我在屏幕上打印出了每个数据集的头部（最前面的行），看一下变量名和初始数据值。

ggplot 函数将数据集名称和图形属性都看作参数，以此来设置图形中各种具体变量。ggplot 函数与 matplotlib 中的 figure 函数很像，都是先收集绘图信息，但并不实际显示各种图形元素，比如点、线、条形等。下一条绘图命令用于添加几何对象，即向图中添加数据的图形表示。脚本中的 3 个添加几何对象的命令分别向图中添加了直方图、线型图和散点图。其余绘图命令向每张统计图中添加标题、坐标轴范围和标签，以及整体布局和颜色主题。

图 6-7 显示了脚本创建的散点图。在 ggplot 文档中（http://ggplot.yhathq.com/docs/index.html），你可以学习到创建和设置统计图表的更多信息。

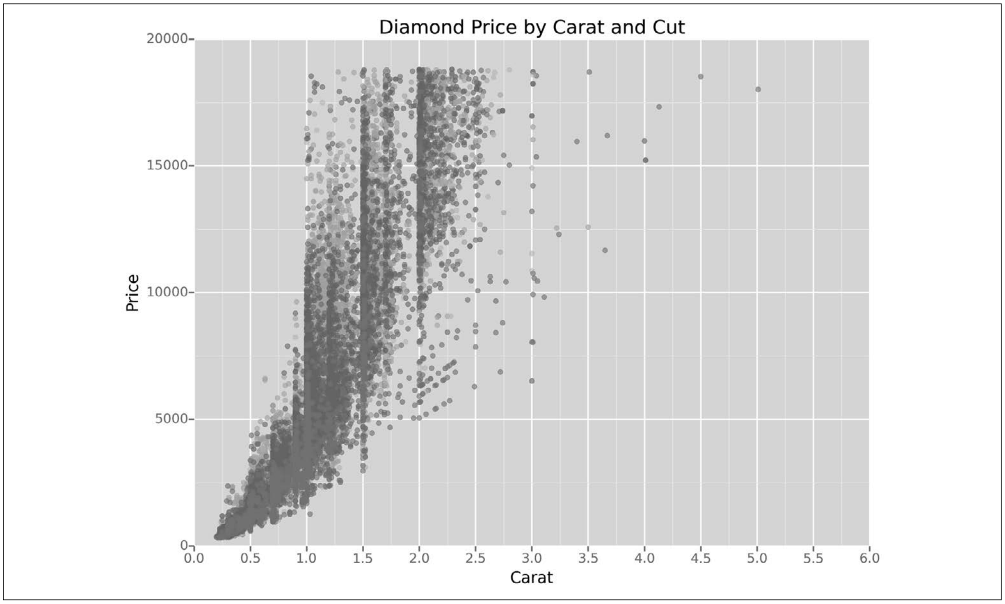


图 6-7：ggplot 生成的散点图

6.4 seaborn

seaborn 简化了在 Python 中创建信息丰富的统计图表的过程。它是在 matplotlib 基础上开发的，支持 numpy 和 pandas 中的数据结构，并集成了 scipy 和 statsmodels 中的统计程序。

seaborn 可以创建标准统计图，包括直方图、密度图、条形图、箱线图和散点图。它可以对成对变量之间的相关性、线性与非线性回归模型以及统计估计的不确定性进行可视化。它可以用来在评估变量时检查变量之间的关系，并可以建立统计图矩阵来显示复杂的关系。它有内置的主题和调色板，可以用来制作精美的图表。最后，因为它是建立在 matplotlib 上的，所以你可以使用 matplotlib 的命令来对图形进行更深入的定制。

下面的脚本 seaborn_plots.py 演示了如何使用 seaborn 创建各种统计图：

```
#!/usr/bin/env python3
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import savefig
sns.set(color_codes=True)
# 直方图
x = np.random.normal(size=100)
sns.distplot(x, bins=20, kde=False, rug=True, label="Histogram w/o Density")
sns.axlabel("Value", "Frequency")
```

```
plt.title("Histogram of a Random Sample from a Normal Distribution")
plt.legend()
# 带有回归直线的散点图与单变量直方图
mean, cov = [5, 10], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
data_frame = pd.DataFrame(data, columns=["x", "y"])
sns.jointplot(x="x", y="y", data=data_frame, kind="reg")\
.set_axis_labels("x", "y")
plt.suptitle("Joint Plot of Two Variables with Bivariate and Univariate Graphs")
# 成对变量之间的散点图与单变量直方图
iris = sns.load_dataset("iris")
sns.pairplot(iris)
# 按照某几个变量生成的箱线图
tips = sns.load_dataset("tips")
sns.factorplot(x="time", y="total_bill", hue="smoker",\
               col="day", data=tips, kind="box", size=4, aspect=.5)
# 带有bootstrap置信区间的线性回归模型
sns.lmplot(x="total_bill", y="tip", data=tips)
# 带有bootstrap置信区间的逻辑斯蒂回归模型
seaborn
tips["big_tip"] = (tips.tip / tips.total_bill) > .15
sns.lmplot(x="total_bill", y="big_tip", data=tips, logistic=True, y_jitter=.03)\
.set_axis_labels("Total Bill", "Big Tip")
plt.title("Logistic Regression of Big Tip vs. Total Bill")
plt.show()
savefig("seaborn_plots.png")
```

第一张统计图是使用 distplot 函数显示的一张直方图，如图 6-8 所示。这个示例中演示的设置包括数据封箱个数、是否显示高斯核密度估计（kde）、在支撑轴上显示地毯图、创建坐标轴标签和标题，以及为图例生成的标签。

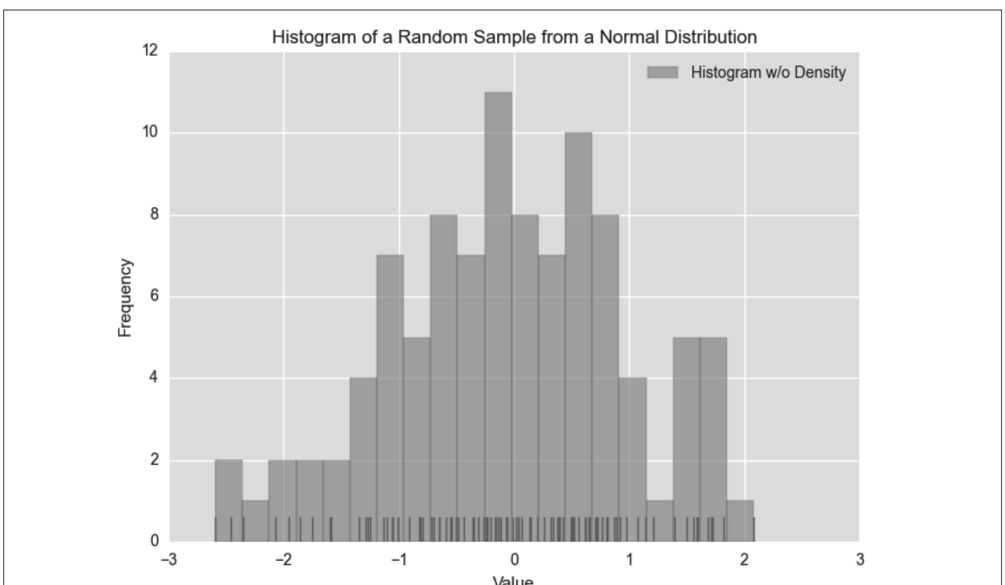


图 6-8：seaborn 生成的直方图，数据来自于对正态分布的随机抽样

第二张统计图是使用 `jointplot` 函数显示两个变量的一张散点图，如图 6-9 所示，其中带有一条回归直线，并为每个变量生成一张直方图。

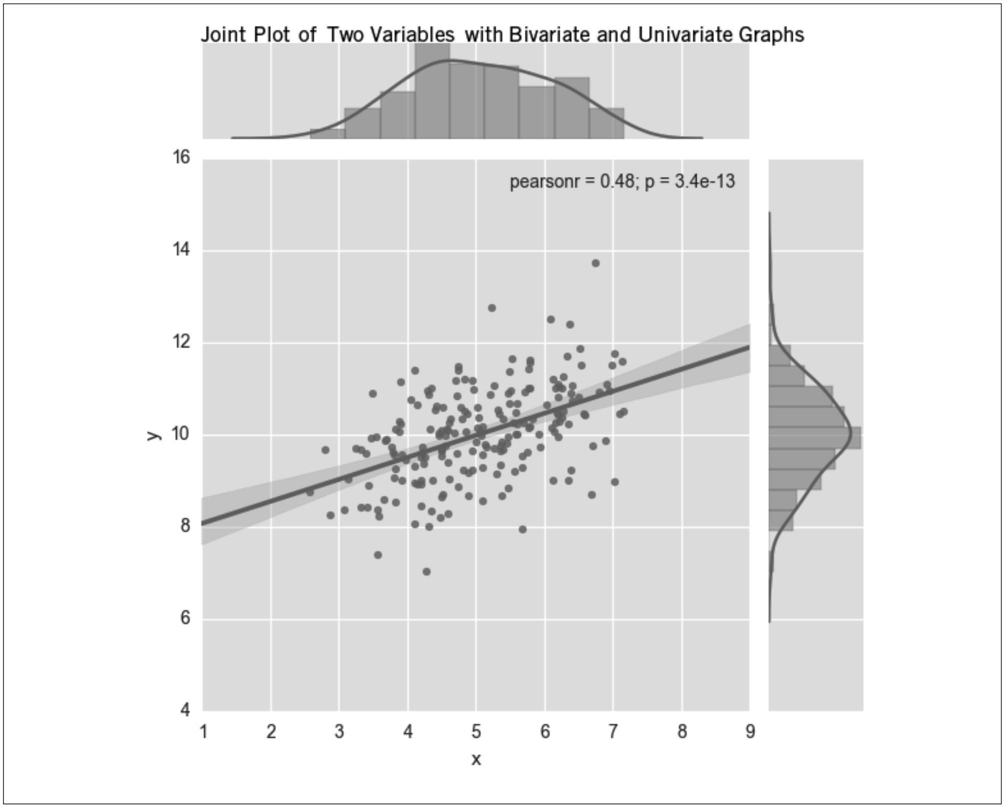


图 6-9：`seaborn` 生成的带有回归直线的散点图，以及两张直方图

第三张统计图是使用 `pairplot` 函数生成数据集中每两个变量之间的双变量散点图，并为每个变量生成一张直方图，如图 6-10 所示。

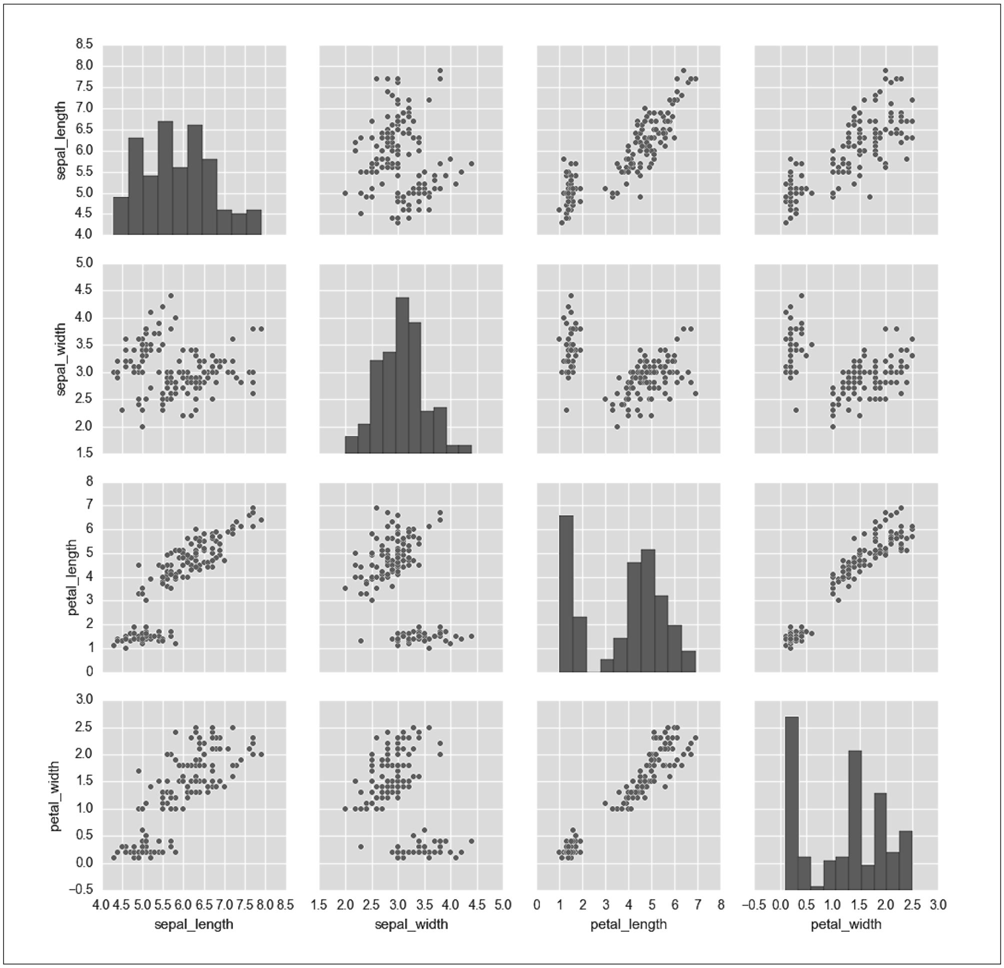


图 6-10：seaborn 生成的鸢尾花数据集中的双变量散点图和单变量直方图

第四张统计图是使用 factorplot 函数生成的箱线图，表示两个变量之间的关系，对于第三个变量的每一个值都有一张箱线图，并按另一个变量分类，如图 6-11 所示。

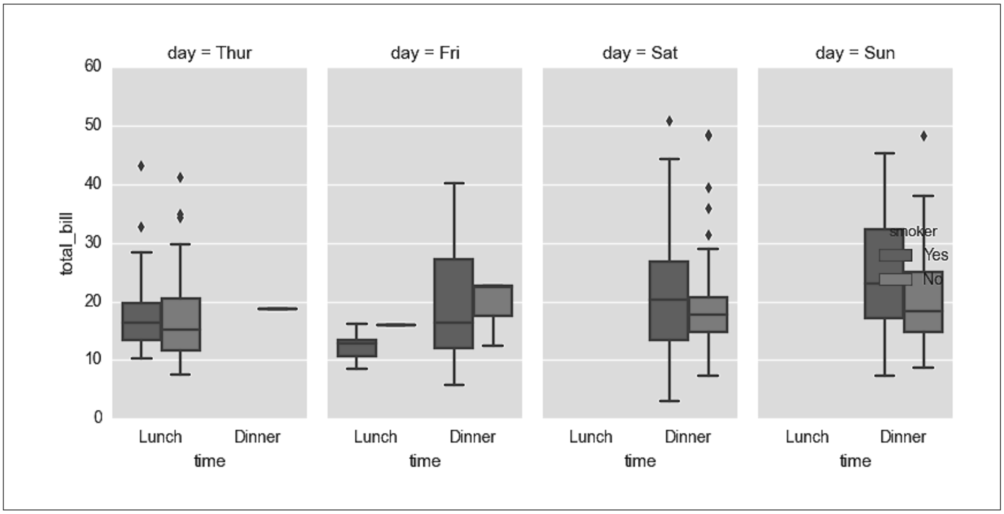


图 6-11：seaborn 生成的账单总数箱线图，按照星期几、每天中的时间和是否吸烟者进行分类

第五张统计图是使用 lmplot 函数生成的一张散点图和一个线性回归模型，并在回归直线周围显示了置信区间，如图 6-12 所示。

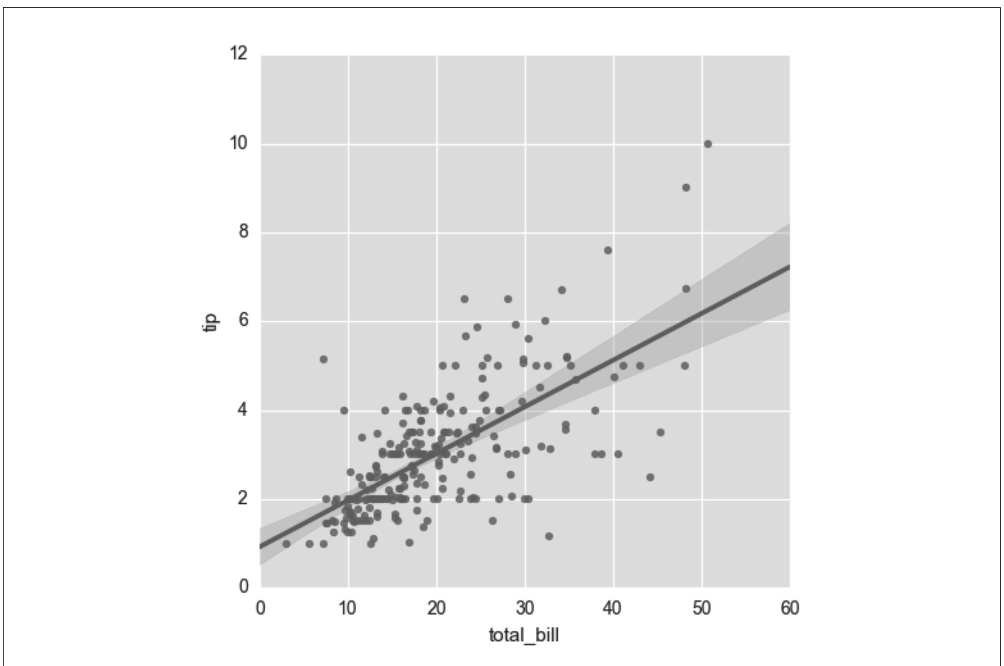


图 6-12：seaborn 生成的散点图和回归直线，表示小费数量和账单总数之间的关系

第六张统计图是使用 `lmplot` 函数为一个二值因变量生成的一个逻辑斯蒂回归模型，如图 6-13 所示。函数使用 `y_jitter` 参数使数据点在 1 和 0 处轻微振动，这样可以更清楚地看出数据点沿着 *x* 轴的聚集情况。

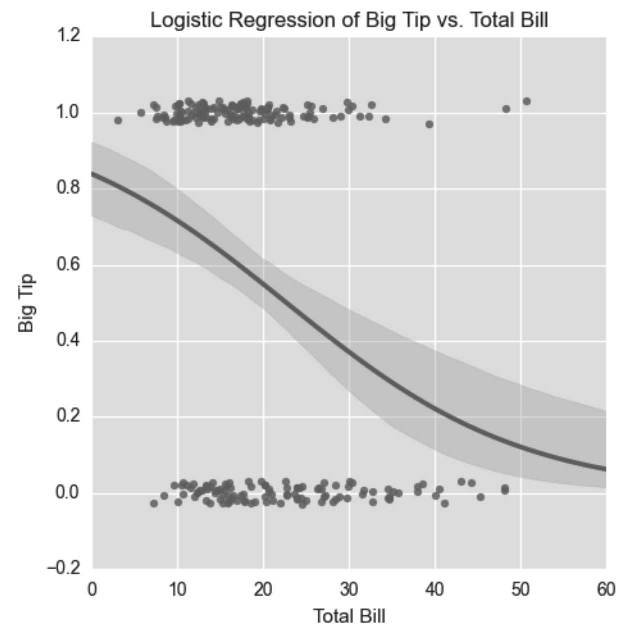


图 6-13：`seaborn` 生成的逻辑斯蒂回归曲线，表示大额小费和账单总额之间的关系

这些示例让你知道了使用 `seaborn` 可以生成何种类型的统计图，但它们只是 `seaborn` 强大功能的冰山一角。你可以参考 `seaborn` 文档（http://stanford.edu/~mwaskom/software/seaborn/index.html），了解创建和设置统计图表的更多信息。

第 7 章

描述性统计与建模

本书前面的章节重点介绍了各种数据处理技术，使用这些技术，我们可以将原始数据转换为可供统计分析的数据集。本章将把注意力放在基本的统计分析和建模技术上，重点介绍如何使用统计图和摘要统计量对数据集进行探索和摘要分析，以及如何使用多元线性回归和逻辑斯蒂回归进行回归和分类分析。

本章并不是对统计分析技术和 `pandas` 功能的综合运用，相反，其目标是演示如何使用 `pandas` 和 `statsmodels` 生成标准的描述性统计量和模型。

7.1　数据集

创建具有成千上万行数据的数据集，不需从零开始，从互联网上下载即可。我们要使用的第一个数据集是葡萄酒质量数据集，从 UCI 机器学习资料库中可以找到。第二个数据集是客户流失数据集，来自于几个数据分析博客。

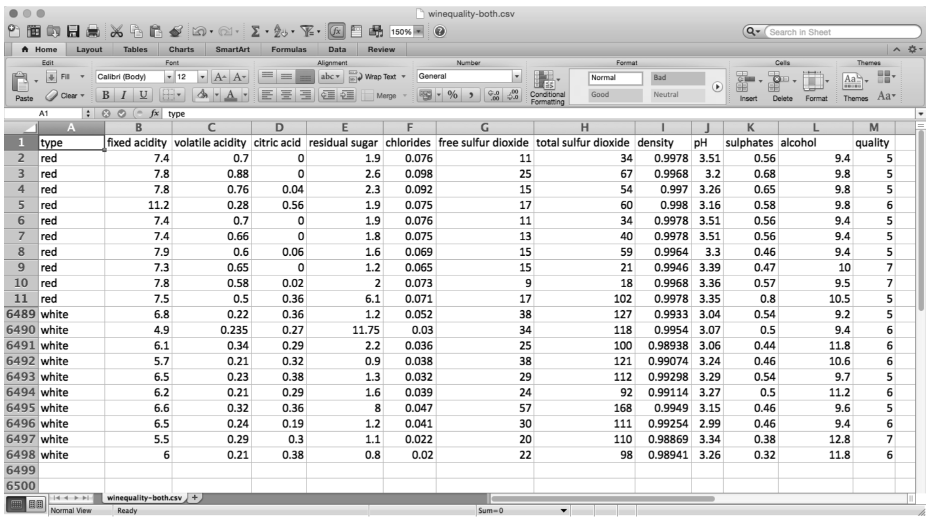
7.1.1　葡萄酒质量

葡萄酒质量数据集包括两个文件，一个是红葡萄酒数据文件，另一个是白葡萄酒数据文件，白葡萄酒是著名的葡萄牙“Vinho Verde”葡萄酒的一个变种。红葡萄酒文件中包含 1599 条观测，白葡萄酒文件中包含 4898 条观测。两个文件中都有 1 个输出变量和 11 个输入变量。输出变量是酒的质量，是一个从 0（低质量）到 10（高质量）的评分。输入变量是葡萄酒的物理化学成分和特性，包括非挥发性酸、挥发性酸、柠檬酸、残余糖分、氯化物、游离二氧化硫、总二氧化硫、密度、pH 值、硫酸盐和酒精含量。

这两个数据集可以通过以下的 URL 下载：

- 红葡萄酒（http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv）
- 白葡萄酒（http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv）

我们不对这两个数据集分别进行分析，而是将它们合成了一个数据集。当你将红葡萄酒数据和白葡萄酒数据合成一个文件后，结果数据集中应该包括一个标题行和 6497 条观测。另外，还应该再添加一列，用来区分这行数据是红葡萄酒还是白葡萄酒的数据。我们要使用的数据集如图 7-1 所示（请注意左侧的行号和第 A 列中新加的“type”变量）。



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
2	red	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
3	red	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5
4	red	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5
5	red	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6
6	red	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7	red	7.4	0.66	0	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
8	red	7.9	0.6	0.06	1.6	0.069	15	59	0.9964	3.3	0.46	9.4	5
9	red	7.3	0.65	0	1.2	0.065	15	21	0.9946	3.39	0.47	10	7
10	red	7.8	0.58	0.02	2	0.073	9	18	0.9968	3.36	0.57	9.5	7
11	red	7.5	0.5	0.36	6.1	0.071	17	102	0.9978	3.35	0.8	10.5	5
6489	white	6.8	0.22	0.36	1.2	0.052	38	127	0.9933	3.04	0.54	9.2	5
6490	white	4.9	0.235	0.27	11.75	0.03	34	118	0.9954	3.07	0.5	9.4	6
6491	white	6.1	0.34	0.29	2.2	0.036	25	100	0.98938	3.06	0.44	11.8	6
6492	white	5.7	0.21	0.32	0.9	0.038	38	121	0.99074	3.24	0.46	10.6	6
6493	white	6.5	0.23	0.38	1.3	0.032	29	112	0.99298	3.29	0.54	9.7	5
6494	white	6.2	0.21	0.29	1.6	0.039	24	92	0.99114	3.27	0.5	11.2	6
6495	white	6.6	0.32	0.36	8	0.047	57	168	0.9949	3.15	0.46	9.6	5
6496	white	6.5	0.24	0.19	1.2	0.041	30	111	0.99254	2.99	0.46	9.4	6
6497	white	5.5	0.29	0.3	1.1	0.022	20	110	0.98869	3.34	0.38	12.8	7
6498	white	6	0.21	0.38	0.8	0.02	22	98	0.98941	3.26	0.32	11.8	6
6499													
6500													

图 7-1：将红葡萄酒数据和白葡萄酒数据连接后的数据集，新增一列 type，表示这行数据来自于哪个数据集

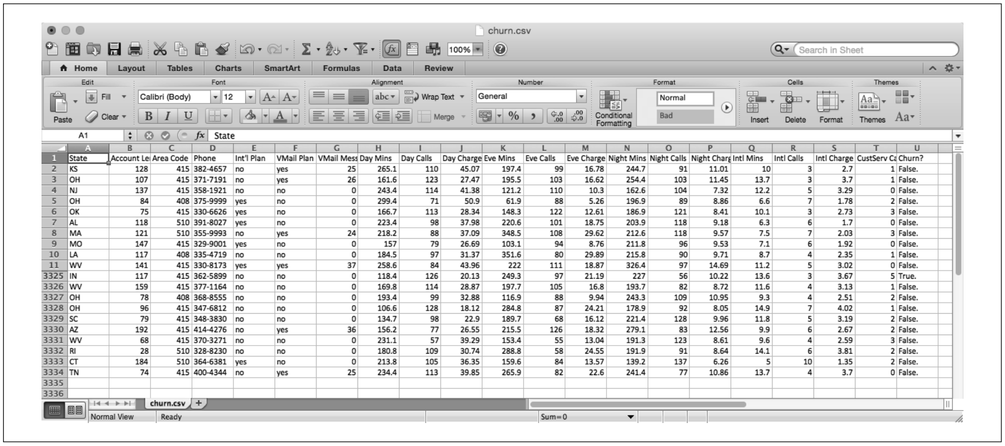
7.1.2　客户流失

客户流失数据集是一个包含 3333 条观测的文件，其中的观测是电信公司现有的和曾经的客户。这个文件有 1 个输出变量和 20 个输入变量。输出变量 `Churn?` 是一个布尔型变量（`True/False`），表示在数据收集的时候，客户是否已经流失（是否还是电信公司的客户）。

输入变量是客户的电话计划和通话行为的特征，包括状态、账户时间、区号、电话号码、是否有国际通话计划、是否有语音信箱、语音信箱消息数量、白天通话时长、白天通话次数、白天通话费用、傍晚通话时长、傍晚通话次数、傍晚通话费用、夜间通话时长、夜间通话次数、夜间通话费用、国际通话时长、国际通话次数、国际通话费用和客户服务通话次数。

这个数据集可以在 Churn（https://raw.githubusercontent.com/EricChiang/churn/master/data/churn.csv）下载。

客户流失数据集如图 7-2 所示。



	State	Account Le	Area Code	Phone	Int'l Plan	VMail Plan	VMail Mess	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charg	Intl Mins	Intl Calls	Intl Charge	CustServ Ca	Churn?
2	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10	3	2.7	1	False.
3	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.7	1	False.
4	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110	10.3	162.6	104	7.32	12.2	5	3.29	0	False.
5	OH	84	408	375-9999	yes	no	0	299.4	71	50.9	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False.
6	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False.
7	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3	6	1.7	0	False.
8	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	7	2.03	3	False.
9	MO	147	415	329-9001	yes	no	0	157	79	26.69	103.1	94	8.76	211.8	96	9.53	7.1	6	1.92	0	False.
10	LA	117	408	335-4719	no	no	0	184.5	97	31.37	351.6	80	29.89	215.8	90	9.71	8.7	4	2.35	1	False.
11	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	222	111	18.87	326.4	97	14.69	11.2	5	3.02	0	False.
3325	IN	117	415	362-5899	no	no	0	118.4	126	20.13	249.3	97	21.19	227	56	10.22	13.6	3	3.67	5	True.
3326	WV	159	415	377-1164	no	no	0	169.8	114	28.87	197.7	105	16.8	193.7	82	8.72	11.6	4	3.13	1	False.
3327	OH	78	408	368-8555	no	no	0	193.4	99	32.88	116.9	88	9.94	243.3	109	10.95	9.3	4	2.51	2	False.
3328	OH	96	415	347-6812	no	no	0	106.6	128	18.12	284.8	87	24.21	178.9	92	8.05	14.9	7	4.02	1	False.
3329	SC	79	415	348-3830	no	no	0	134.7	98	22.9	189.7	68	16.12	221.4	128	9.96	11.8	5	3.19	2	False.
3330	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	215.5	126	18.32	279.1	83	12.56	9.9	6	2.67	2	False.
3331	WV	68	415	370-3271	no	no	0	231.1	57	39.29	153.4	55	13.04	191.3	123	8.61	9.6	4	2.59	3	False.
3332	RI	28	510	328-8230	no	no	0	180.8	109	30.74	288.8	58	24.55	191.9	91	8.64	14.1	6	3.81	2	False.
3333	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	159.6	84	13.57	139.2	137	6.26	5	10	1.35	2	False.
3334	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	265.9	82	22.6	241.4	77	10.86	13.7	4	3.7	0	False.

图 7-2：客户流失数据集的头部和尾部

7.2 葡萄酒质量

7.2.1 描述性统计

下面先来分析葡萄酒质量数据集。首先，计算出每列的总体描述性统计量、质量列中的唯一值以及和这个唯一值对应的观测数量。要完成这个任务，需要创建一个新脚本 wine_quality.py，然后输入以下代码：

```
#!/usr/bin/env python3
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.formula.api import ols, glm
# 将数据集读入到pandas数据框中
wine = pd.read_csv('winequality-both.csv', sep=',', header=0)
wine.columns = wine.columns.str.replace(' ', '_')
print(wine.head())
# 显示所有变量的描述性统计量
print(wine.describe())
# 找出唯一值
print(sorted(wine.quality.unique()))
# 计算值的频率
print(wine.quality.value_counts())
```

在 import 语句之后，要做的第一件事就是使用 pandas 的 read_csv 方法将文本文件 winequality-both.csv 读入一个 pandas 数据框。附加参数表示域分隔符为逗号，第一行为列

标题。有些列标题中包含空格（例如：fixed acidity），所以在下面一行代码中要使用下划线替换空格。然后，使用 head 函数检查一下标题行和前 5 行数据，确保数据被正确加载。

第 14 行代码使用 pandas 的 describe 函数打印出数据集中每个数值型变量的摘要统计量。这些统计量包括总数、均值、标准差、最小值、第 25 个百分位数、中位数、第 75 个百分位数和最大值。例如，质量评分中有 6497 个观测，评分范围从 3 到 9，平均质量评分为 5.8，标准差为 0.87。

下面一行代码识别出质量列中的唯一值，并以升序打印在屏幕上。输出显示质量列中的唯一值是 3、4、5、6、7、8 和 9。

最终，本节最后一行代码计算出质量列中每个唯一值在数据集中出现的次数，并把它们以降序打印到屏幕上。输出显示，有 2836 个观测的质量评分为 6，2138 个观测的质量评分为 5，1079 个观测的质量评分为 7，216 个观测的质量评分为 4，193 个观测的质量评分为 8，30 个观测的质量评分为 3，5 个观测的质量评分为 9。

7.2.2　分组、直方图与*t*检验

前面计算出的统计量是针对整个数据集的，既包括红葡萄酒数据也包括白葡萄酒数据。下面我们分别分析红葡萄酒数据和白葡萄酒数据，看看统计量是否会保持不变：

```
# 按照葡萄酒类型显示质量的描述性统计量
print(wine.groupby('type')[['quality']].describe().unstack('type'))
# 按照葡萄酒类型显示质量的特定分位数值
print(wine.groupby('type')[['quality']].quantile([0.25, 0.75]).unstack('type'))
# 按照葡萄酒类型查看质量分布
red_wine = wine.loc[wine['type']=='red', 'quality']
white_wine = wine.loc[wine['type']=='white', 'quality']
sns.set_style("dark")
print(sns.distplot(red_wine, \
        norm_hist=True, kde=False, color="red", label="Red wine"))
print(sns.distplot(white_wine, \
        norm_hist=True, kde=False, color="white", label="White wine"))
sns.axlabel("Quality Score", "Density")
plt.title("Distribution of Quality by Wine Type")
plt.legend()
plt.show()
# 检验红葡萄酒和白葡萄酒的平均质量是否有所不同
print(wine.groupby(['type'])[['quality']].agg(['std']))
tstat, pvalue, df = sm.stats.ttest_ind(red_wine, white_wine)
print('tstat: %.3f pvalue: %.4f' % (tstat, pvalue))
```

本节的第一行代码分别打印出红葡萄酒和白葡萄酒的摘要统计量。groupby 函数使用 type 列中的两个值将数据分为两组。方括号可以生成一个列表，列表中的元素是用来生成输出的列。在本例中，只对质量列应用 describe 函数。这些命令的结果就是生成一列统计量，来自红葡萄酒数据的计算结果和白葡萄酒数据的计算结果是相互垂直地堆叠在一起的。unstack 函数将结果重新排列，这样红葡萄酒和白葡萄酒的统计量就会显示在并排的两列中。

下一行代码和它前面的代码非常相似，但不是使用 `describe` 函数计算一些描述性统计量，而是使用 `quantile` 函数对质量列计算第 25 百分位数和第 75 百分位数。

接下来，使用 `seaborn` 创建一幅统计图，其中有两个直方图，一个表示红葡萄酒，另一个表示白葡萄酒，如图 7-3 所示。

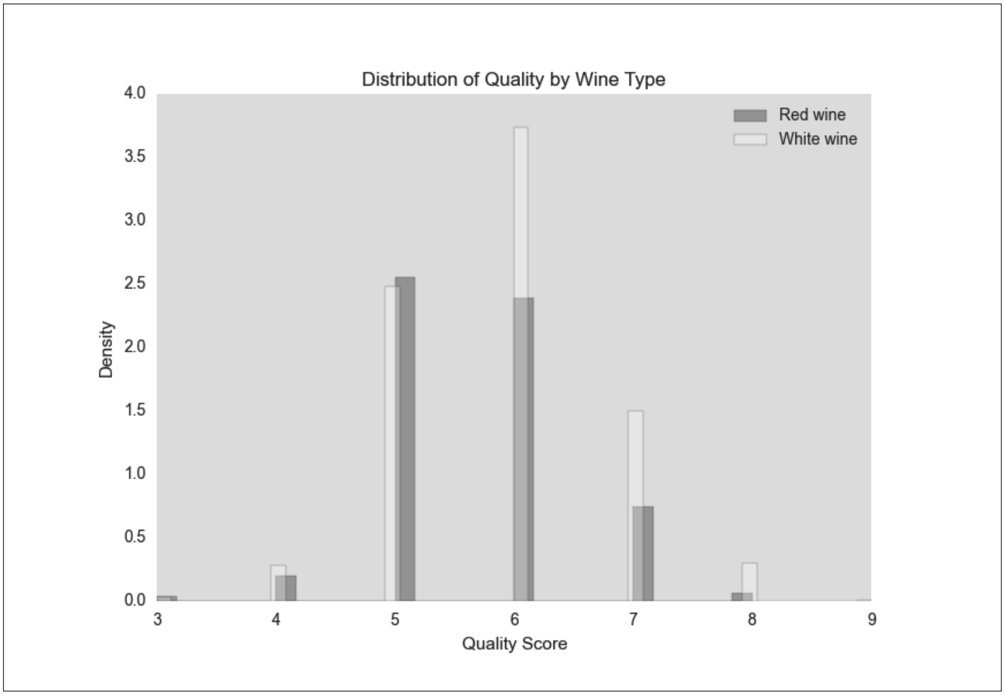


图 7-3：表示两种类型葡萄酒评分分布的密度直方图

红条表示红葡萄酒，白条表示白葡萄酒。因为白葡萄酒数据比红葡萄酒多（白葡萄酒有 4898 条数据，红葡萄酒有 1599 条数据），所以图中显示密度分布，不显示频率分布。从这个统计图可以看出，两种葡萄酒的评分都近似正态分布。与原始数据的摘要统计量相比，直方图更容易看出两种葡萄酒的质量评分的分布。

最后，进行一下 t 检验，判断红葡萄酒和白葡萄酒的平均评分是否有区别。这段代码演示了如何使用 `groupby` 和 `agg` 函数来为数据集中的分组计算一系列统计量。在本例中，我们想知道红葡萄酒和白葡萄酒评分的标准差是否相同，所以在 t 检验中可以使用合并方差。t 检验统计量为 −9.69，p 值为 0.00，这说明白葡萄酒的平均质量评分在统计意义上大于红葡萄酒的平均质量评分。

7.2.3　成对变量之间的关系和相关性

前面已经检查了输出变量，下面简单研究一下输入变量。让我们计算一下输入变量两两之间的相关性，并为一些输入变量创建带有回归直线的散点图：

```
# 计算所有变量的相关矩阵
print(wine.corr())
# 从红葡萄酒和白葡萄酒的数据中取出一个“小”样本来进行绘图
def take_sample(data_frame, replace=False, n=200):
    return data_frame.loc[np.random.choice(data_frame.index, \
    replace=replace, size=n)]
reds_sample = take_sample(wine.loc[wine['type']=='red', :])
whites_sample = take_sample(wine.loc[wine['type']=='white', :])
wine_sample = pd.concat([reds_sample, whites_sample])
wine['in_sample'] = np.where(wine.index.isin(wine_sample.index), 1.,0.)
print(pd.crosstab(wine.in_sample, wine.type, margins=True))
# 查看成对变量之间的关系
sns.set_style("dark")
g = sns.pairplot(wine_sample, kind='reg', plot_kws={"ci": False,\
"x_jitter": 0.25, "y_jitter": 0.25}, hue='type', diag_kind='hist',\
diag_kws={"bins": 10, "alpha": 1.0}, palette=dict(red="red", white="white"),\
markers=["o", "s"], vars=['quality', 'alcohol', 'residual_sugar'])
print(g)
plt.suptitle('Histograms and Scatter Plots of Quality, Alcohol, and Residual\
Sugar', fontsize=14, horizontalalignment='center', verticalalignment='top',\
x=0.5, y=0.999)
plt.show()
```

corr 函数可以计算出数据集中所有变量两两之间的线性相关性。根据相关系数的符号，从输出中可以知道酒精含量、硫酸盐、pH 值、游离二氧化硫和柠檬酸这些指标与质量是正相关的，相反，非挥发性酸、挥发性酸、残余糖分、氯化物、总二氧化硫和密度这些指标与质量是负相关的。

数据集中有 6000 多个点，所以如果将它们都画在统计图中，就很难分辨出清楚的点。为解决这个问题，我们定义了一个函数 take_sample，用来抽取在统计图中使用的样本点。这个函数使用 pandas 数据框索引和 numpy 的 random.choice 函数随机选择一个行的子集。我们使用这个函数对红葡萄酒和白葡萄酒分别进行抽样，并将抽样所得的两个数据框连接成一个数据框。然后，在 wine 数据框中创建一个新列 in_sample，并使用 numpy 的 where 函数和 pandas 的 isin 函数对这个新列进行填充，填充的值根据此行的索引值是否在抽样数据的索引值中分别设为 1 和 0。最后，我们使用 pandas 的 crosstab 函数来确认 in_sample 列中包含 400 个 1（200 条红葡萄酒数据和 200 条白葡萄酒数据）和 6097 个 0。

seaborn 的 pairplot 函数可以创建一个统计图矩阵。主对角线上的图以直方图或密度图的形式显示了每个变量的单变量分布，对角线之外的图以散点图的形式显示了每两个变量之间的双变量分布，散点图中可以有回归直线，也可以没有。

图 7-4 中的统计图显示了葡萄酒质量、酒精含量和残余糖分之间的关系。红条和红点表示红葡萄酒，白条和白点表示白葡萄酒。因为质量评分都是整数，所以我加上了一点振动，这样更容易看出数据在何处集中。

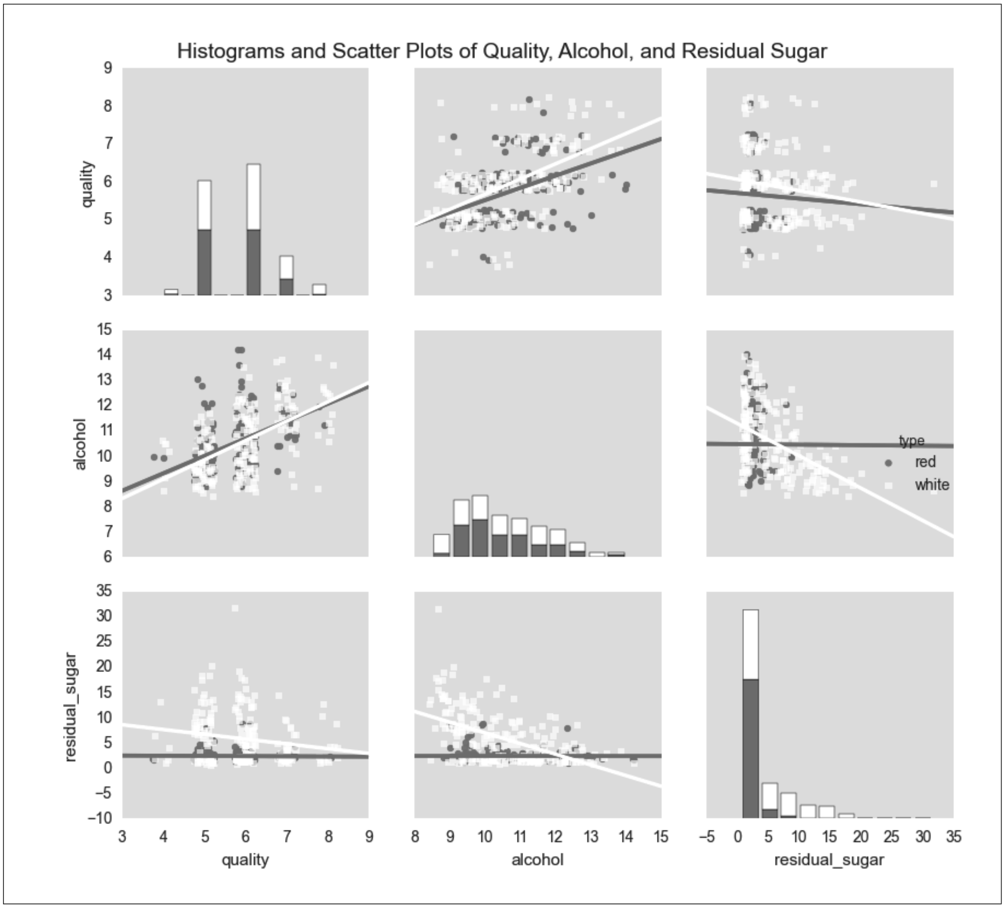


图 7-4：质量、酒精含量和残余糖分这 3 个变量两两之间的散点图、回归直线和直方图，按葡萄酒类型分类

从这些统计图可以看出，对于红葡萄酒和白葡萄酒来说，酒精含量的均值和标准差是大致相同的，但是，白葡萄酒残余糖分的均值和标准差却大于红葡萄酒残余糖分的均值和标准差。从回归直线可以看出，对于两种类型的葡萄酒，酒精含量增加时，质量评分也随之提高，相反，残余糖分增加时，质量评分则随之降低。这两个变量对白葡萄酒的影响都要大于对红葡萄酒的影响。

7.2.4　使用最小二乘估计进行线性回归

相关系数和两两变量之间的统计图有助于对两个变量之间的关系进行量化和可视化，但是它们不能测量出每个自变量在其他自变量不变时与因变量之间的关系。线性回归可以解决这个问题。

线性回归模型如下：

- $y_i \sim N(\mu_i, \sigma^2)$,
- $\mu_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}$

对于 $i = 1, 2, \cdots, n$ 个观测和 p 个自变量。

这个模型表示观测 y_i 服从均值为 μ_i 方差为 σ^2 的正态分布（高斯分布），其中 μ_i 依赖于自变量，σ^2 为一个常数。也就是说，给定了自变量的值之后，我们就可以得到一个具体的质量评分，但在另一天，给定同样的自变量值，我们可能会得到一个和前面不同的质量评分。但是，经过很多天自变量取同样的值（也就是一个很长的周期），质量评分会落在 $\mu_i \pm \sigma$ 这个范围内。

理解了线性回归模型后，下面让我们使用 statsmodel 包来进行线性回归：

```
my_formula = 'quality ~ alcohol + chlorides + citric_acid + density\
+ fixed_acidity + free_sulfur_dioxide + pH + residual_sugar + sulphates\
+ total_sulfur_dioxide + volatile_acidity'

lm = ols(my_formula, data=wine).fit()

## 或者,也可以使用广义线性模型(glm)语法进行线性回归
## lm = glm(my_formula, data=wine, family=sm.families.Gaussian()).fit()

print(lm.summary())
print("\nQuantities you can extract from the result:\n%s" % dir(lm))
print("\nCoefficients:\n%s" % lm.params)
print("\nCoefficient Std Errors:\n%s" % lm.bse)
print("\nAdj. R-squared:\n%.2f" % lm.rsquared_adj)
print("\nF-statistic: %.1f   P-value: %.2f" % (lm.fvalue, lm.f_pvalue))
print("\nNumber of obs: %d   Number of fitted values: %d" % (lm.nobs,\
len(lm.fittedvalues)))
```

第一行代码将一个字符串赋给变量 my_foumula。这个字符串中包含的是类似 R 语言语法的回归公式定义。波浪线（~）左侧的变量 quality 是因变量，波浪线右侧的变量是自变量。

第二行代码使用公式和数据拟合一个普通最小二乘回归模型，并将结果赋给变量 lm。为了演示另外一种拟合方法，下一行代码（注释中的代码）使用广义线性模型（glm）的语法代替普通最小二乘语法，拟合同样的模型。

下面 7 行代码向屏幕上打印模型结果。第一行向屏幕上打印结果的摘要信息。这些摘要信息非常重要，因为它包含了模型系数、系数的标准差和置信区间、修正 R 方、F 统计量等模型详细信息。

下一行代码打印出一个列表，其中包含从模型对象 lm 中提取出的所有数值信息，检查了这个列表之后，我希望提取出模型系数、系数的标准差、修正 R 方、F 统计量和它的 p 值，以及模型拟合值。

接下来的 4 行代码提取出了这些值。lm.params 以一个序列的形式返回模型系数，这样你可以通过定位或名称提取出单个的系数。例如，要提取酒精含量的系数 0.267，你可以使用 lm.params[1] 或 lm.params['alcohol']。同样，lm.bse 以序列的形式返回模型系数的标准差。lm.rsquared_adj 返回修正 R 方，lm.fvalue 和 lm.f_pvalue 分别返回 F 统计量和它

的 p 值。最后，`lm.fittedvalues` 返回拟合值。我没有给出所有的拟合值，而是在观测总数 `lm.nobs` 后面显示拟合值的数量，以确认拟合值与观测具有同样的数量。输出结果如图 7-5 所示。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                quality   R-squared:                       0.292
Model:                            OLS   Adj. R-squared:                  0.291
Method:                 Least Squares   F-statistic:                     243.3
Date:                Sat, 28 May 2016   Prob (F-statistic):               0.00
Time:                        09:20:12   Log-Likelihood:                -7215.5
No. Observations:                6497   AIC:                         1.445e+04
Df Residuals:                    6485   BIC:                         1.454e+04
Df Model:                          11
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------------
Intercept               55.7627     11.894      4.688      0.000        32.447    79.079
alcohol                  0.2670      0.017     15.963      0.000         0.234     0.300
chlorides               -0.4837      0.333     -1.454      0.146        -1.136     0.168
citric_acid             -0.1097      0.080     -1.377      0.168        -0.266     0.046
density                -54.9669     12.137     -4.529      0.000       -78.760   -31.173
fixed_acidity            0.0677      0.016      4.346      0.000         0.037     0.098
free_sulfur_dioxide      0.0060      0.001      7.948      0.000         0.004     0.007
pH                       0.4393      0.090      4.861      0.000         0.262     0.616
residual_sugar           0.0436      0.005      8.449      0.000         0.033     0.054
sulphates                0.7683      0.076     10.092      0.000         0.619     0.917
total_sulfur_dioxide    -0.0025      0.000     -8.969      0.000        -0.003    -0.002
volatile_acidity        -1.3279      0.077    -17.162      0.000        -1.480    -1.176
==============================================================================
Omnibus:                      144.075   Durbin-Watson:                   1.646
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              324.712
Skew:                          -0.006   Prob(JB):                     3.09e-71
Kurtosis:                       4.095   Cond. No.                     2.49e+05
==============================================================================
```

图 7-5：具有 11 个特征的葡萄酒质量多元线性回归

7.2.5　系数解释

如果你想使用这个模型弄清楚因变量（葡萄酒质量）和自变量（11 个葡萄酒特性）之间的关系，就应该解释一下模型系数的意义。在这个模型中，某个自变量系数的意义是，在其他自变量保持不变的情况下，这个自变量发生 1 个单位的变化时，导致葡萄酒质量评分发生的平均变化。例如，酒精含量系数的含义就是，从平均意义上来说，如果两种葡萄酒其他自变量的值都相同，那么酒精含量高 1 个单位的葡萄酒的质量评分就要比另一种葡萄酒的质量评分高出 0.27 分。

并不是所有的系数都需要解释。例如，截距系数的意义是当所有自变量的值都为 0 时的期望评分。因为没有任何一种葡萄酒的各种成分都为 0，所以截距系数没有具体意义。

7.2.6　自变量标准化

关于这个模型，还需要注意的一点是，普通最小二乘回归是通过使残差平方和最小化来估计未知的 β 参数值的，这里的残差是指自变量观测值与拟合值之间的差别。因为残差大小是依赖于自变量的测量单位的，所以如果自变量的测量单位相差很大的话，那么将自变量标准化后，就可以更容易对模型进行解释了。对自变量进行标准化的方法是，先从自变量的每个观测值中减去均值，然后再除以这个自变量的标准差。自变量标准化完

成以后，它的均值为 0，标准差为 1[1]。

使用 wine.describe()，可以看到氯化物的范围是从 0.009 到 0.661，而总二氧化硫的范围是从 6.0 到 440.0。其余各变量的最小值与最大值之间的区别也大致如此。因为各个自变量值的变化范围相差非常悬殊，所以非常应该对自变量进行标准化，看看这样做了之后，能否更容易对结果进行解释。

pandas 在数据框中对变量进行标准化非常容易。你可以对一个观测写一个变换公式，pandas 可以把这个公式扩展到行与列中，来标准化所有变量。以下各行代码使用标准化后的自变量创建了一个新数据框 wine_standardized：

```
# 创建一个名为dependent_variable的序列来保存质量数据
dependent_variable = wine['quality']

# 创建一个名为independent variables的数据框
# 来保存初始的葡萄酒数据集中除quality、type和in_sample之外的所有变量
independent_variables = wine[wine.columns.difference(['quality', 'type',\
'in_sample'])]

# 对自变量进行标准化
# 对每个变量,在每个观测中减去变量的均值
# 并且使用结果除以变量的标准差
independent_variables_standardized = (independent_variables -\
independent_variables.mean()) / independent_variables.std()

# 将因变量quality作为一列添加到自变量数据框中
# 创建一个带有标准化自变量的
# 新数据集
wine_standardized = pd.concat([dependent_variable, independent_variables\
_standardized], axis=1)
```

完成了带有标准化自变量的数据集之后，让我们重新进行线性回归，并查看一下摘要统计（见图 7-6）：

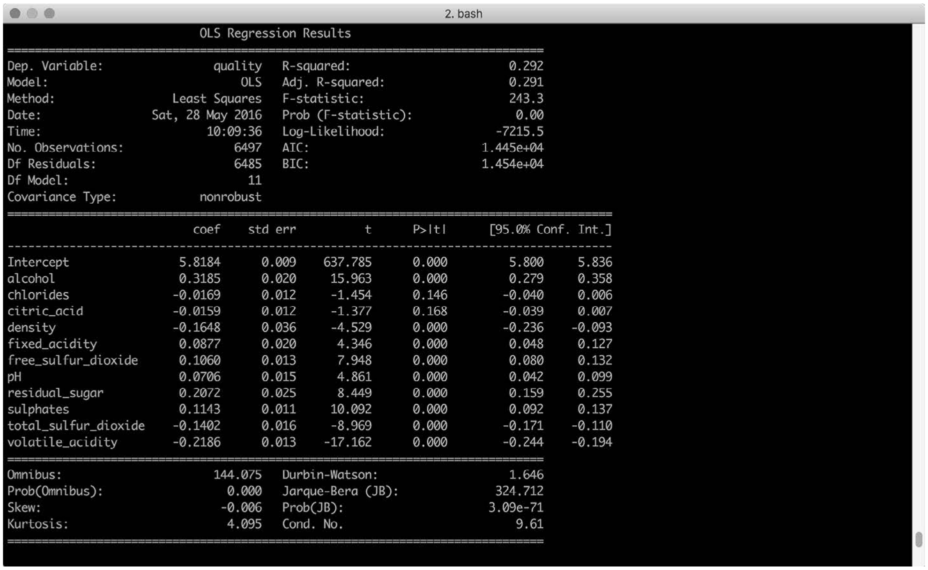
```
lm_standardized = ols(my_formula, data=wine_standardized).fit()
print(lm_standardized.summary())
```

自变量标准化会改变我们对模型系数的解释。现在每个自变量系数的含义是，不同的葡萄酒在其他自变量均相同的情况下，某个自变量相差 1 个标准差，会使葡萄酒的质量评分平均相差多少个标准差。举个例子，酒精含量系数的意义是，从平均意义上说，如果两种葡萄酒其他自变量的值都相同，那么酒精含量高 1 个标准差的葡萄酒的质量评分就要比另一种葡萄酒的质量评分高出 0.32 个标准差。

还是通过 wine.describe() 函数，我们可以看到酒精含量的均值和标准差是 10.5 和 1.2，质

注 1：在 *Data Analysis Using Regression and Multilevel/Hierarchical Models* (Cambridge University Press, 2007, p.56) 中，Gelman and Hill 建议在数据集中既有连续型自变量又有二值型自变量的情况下，用两倍标准差去除，而不是用一倍标准差。这样的话，标准化自变量一个单位的变化就对应于均值上下一个标准差的变化。因为葡萄酒质量数据集中不包含二值自变量，所以我通过除以一倍标准差将自变量标准化为 z-scores。

量评分的均值和标准差是 5.8 和 0.9。因此，从平均意义上说，如果两种葡萄酒其他的自变量值均相同，那么酒精含量为 11.7（10.5+1.2）的葡萄酒的质量评分就会比酒精含量为均值的葡萄酒的质量评分大 0.32 个标准差。



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                quality   R-squared:                       0.292
Model:                            OLS   Adj. R-squared:                  0.291
Method:                 Least Squares   F-statistic:                     243.3
Date:                Sat, 28 May 2016   Prob (F-statistic):               0.00
Time:                        10:09:36   Log-Likelihood:                -7215.5
No. Observations:                6497   AIC:                         1.445e+04
Df Residuals:                    6485   BIC:                         1.454e+04
Df Model:                          11
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------------
Intercept                5.8184      0.009    637.785      0.000         5.800     5.836
alcohol                  0.3185      0.020     15.963      0.000         0.279     0.358
chlorides               -0.0169      0.012     -1.454      0.146        -0.040     0.006
citric_acid             -0.0159      0.012     -1.377      0.168        -0.039     0.007
density                 -0.1648      0.036     -4.529      0.000        -0.236    -0.093
fixed_acidity            0.0877      0.020      4.346      0.000         0.048     0.127
free_sulfur_dioxide      0.1060      0.013      7.948      0.000         0.080     0.132
pH                       0.0706      0.015      4.861      0.000         0.042     0.099
residual_sugar           0.2072      0.025      8.449      0.000         0.159     0.255
sulphates                0.1143      0.011     10.092      0.000         0.092     0.137
total_sulfur_dioxide    -0.1402      0.016     -8.969      0.000        -0.171    -0.110
volatile_acidity        -0.2186      0.013    -17.162      0.000        -0.244    -0.194
==============================================================================
Omnibus:                      144.075   Durbin-Watson:                   1.646
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              324.712
Skew:                          -0.006   Prob(JB):                     3.09e-71
Kurtosis:                       4.095   Cond. No.                         9.61
==============================================================================
```

图 7-6：具有 11 个特征的葡萄酒质量多元线性回归——在回归之前，自变量被标准化为 z-scores

自变量标准化同样会改变我们对截距的解释。当解释变量被标准化后，截距表示的就是当所有自变量取值为均值时因变量的均值。在我们的模型摘要中，截距系数的意义就是当一种葡萄酒所有的成分都取均值的时候，它的质量评分均值应该是 5.8，标准差为 0.009。

7.2.7　预测

在某些情况下，我们需要使用没有用来拟合模型的新数据进行预测。例如，你会收到关于葡萄酒成分的一个新的观测，并需要根据这些成分预测这种葡萄酒的质量评分。让我们通过选择现有数据集的前 10 个观测，并根据它们的葡萄酒成分预测质量评分，来演示一下如何对新数据做出预测。

需要说明的是，仅出于方便和演示的目的，我们才使用这些已经用于拟合模型的数据。除了这个示例之外，你应该使用未用于拟合模型的数据来评价模型，并用新的观测数据进行预测。记住了这一点之后，下面创建一组“新”观测，并使用它们来预测质量评分：

```
# 使用葡萄酒数据集中的前10个观测创建10个“新”观测
# 新观测中只包含模型中使用的自变量
new_observations = wine.ix[wine.index.isin(range(10)), \
independent_variables.columns]

# 基于新观测中的葡萄酒特性预测质量评分
y_predicted = lm.predict(new_observations)
```

```
# 将预测值保留两位小数并打印到屏幕上
y_predicted_rounded = [round(score, 2) for score in y_predicted]
print(y_predicted_rounded)
```

变量 y_predicted 中包含着 10 个预测值。为了使输出更简单易懂，我将预测值保留两位小数。在这个示例中，如果我们使用的是真正的新观测，就可以使用这些预测值来评价模型。无论如何，我们得到了一些预测值，可以用来评估或做些别的事情。

7.3 客户流失

下面来分析客户流失数据集。首先，将数据读入一个数据框，然后格式化列标题，为数据框 churn 创建一个新的数值型二值变量，并检查数据框中前面几行数据。为了完成这些操作，需要创建一个新脚本 customer_churn.py，并输入以下各行代码：

```
#!/usr/bin/env python3
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
churn = pd.read_csv('churn.csv', sep=',', header=0)
churn.columns = [heading.lower() for heading in \
churn.columns.str.replace(' ', '_').str.replace("\'", "").str.strip('?')]
churn['churn01'] = np.where(churn['churn'] == 'True.', 1., 0.)
print(churn.head())
```

import 语句之后的第一行代码将数据读入数据框 churn。下一行代码两次使用 replace 函数将列标题中的空格替换成下划线，并删除嵌入的单引号。请注意第二个 replace 函数，其中第一个参数的两个双引号中间是一个反斜线加一个单引号，逗号后面的第二个参数就是一对双引号。这行代码还使用 strip 函数除去了列标题 Churn? 末尾的问号。最后，这行代码使用列表生成式将所有列标题转换为小写。

下一行代码创建一个新列 churn01，并使用 numpy 的 where 函数根据 churn 这一列中的值用 1 或 0 来填充它。churn 这一列中的值不是 True 就是 False，所以如果 churn 中的值是 True，那么 churn01 中的值就是 1，如果 churn 中的值是 False，那么 churn01 中的值就是 0。

最后一行代码使用 head 函数显示标题行和前 5 个数据行，这样可以检查一下数据加载以及列标题的格式化是否正确。

在将数据加载到数据框中之后，可以通过计算流失客户和未流失客户的描述性统计量，来看看这两组客户有什么区别。下面的代码按照 churn 这一列中的值将数据分成了两组：已流失的客户和未流失的客户。然后为每个分组中的一些特定的列计算 3 个统计量：总数、均值和标准差：

```
# 为分组数据计算描述性统计量
print(churn.groupby(['churn'])[['day_charge', 'eve_charge', 'night_charge',\
'intl_charge', 'account_length', 'custserv_calls']].agg(['count', 'mean',\
'std']))
```

下面的代码演示了如何为不同的变量计算不同类型的多个统计量。这行代码为 4 个变量计算均值和标准差，为 2 个变量计算总数、最小值和最大值。我们还是按照 churn 列分组，所以代码分别为流失客户和未流失客户计算这些统计量：

```
# 为不同的变量计算不同的统计量
print(churn.groupby(['churn']).agg({'day_charge' : ['mean', 'std'],
                   'eve_charge' : ['mean', 'std'],
                   'night_charge' : ['mean', 'std'],
                   'intl_charge' : ['mean', 'std'],
                   'account_length' : ['count', 'min', 'max'],
                   'custserv_calls' : ['count', 'min', 'max']}))
```

下一段代码对客户服务通话次数这部分数据进行了摘要分析，先按照一个新变量 total_charges 中的值使用等宽分箱法将数据分成 5 个组，然后为每个分组计算 5 个统计量：总数、最小值、均值、最大值和标准差。为了完成这些操作，第一行代码创建一个新变量 total_charges，表示白天、傍晚、夜间和国际通话费用的总和。下一行代码使用 cut 函数按照等宽分箱法将 total_charges 分成 5 组。然后定义一个函数 get_stats，为每个分组返回一个统计量字典。下一行代码按照 5 个 total_charges 分组将客户服务通话次数也分成同样的 5 组。最后，在分组数据上应用 get_stats 函数，为 5 个分组计算统计量：

```
# 创建total_charges
# 将其分为5组,并为每一组计算统计量
churn['total_charges'] = churn['day_charge'] +churn['eve_charge'] +\
churn['night_charge'] + churn['intl_charge']
factor_cut = pd.cut(churn.total_charges, 5, precision=2)
def get_stats(group):
    return {'min' : group.min(), 'max' : group.max(),
            'count' : group.count(), 'mean' : group.mean(),
            'std' : group.std()}
grouped = churn.custserv_calls.groupby(factor_cut)
print(grouped.apply(get_stats).unstack())
```

和前一段代码相似，下一段代码也是用 5 个统计量对客户服务通话数据进行了摘要分析。但是，这段代码使用 qcut 函数通过等深分箱法（按照分位数进行划分）将 account_length 分成了 4 组，而不是使用等宽分箱法对数据进行的分组：

```
# 将account_length按照分位数进行分组
# 并为每个分位数分组计算统计量
factor_qcut = pd.qcut(churn.account_length, [0., 0.25, 0.5, 0.75, 1.])
grouped = churn.custserv_calls.groupby(factor_qcut)
print(grouped.apply(get_stats).unstack())
```

通过分位数对 account_length 进行划分，可以保证每个分组中包含数目大致相同的观测。前一段代码中通过等宽分箱法得到的每个分组中包含的观测数目是不一样的。qcut 函数使用一个整数或一个分位数数组来设定分位数的数量，所以你可以使用整数 4 来代替 [0., 0.25, 0.5, 0.75, 1.] 设定 4 等分，或使用 10 来设定 10 等分。

下面一段代码演示了如何使用 pandas 的 get_dummies 函数来创建二值指标变量，并将这些变量添加进数据框。前两行代码为 intl_plan 列和 vmail_plan 列创建二值指标变量，并

使用原来的变量名作为新列的前缀。下一行使用 join 命令将 churn 列和新的二值指标列合并，并将结果赋给一个新的文本框 churn_with_dummies。这个新文本框有 5 列：churn、intl_plan_no、intl_plan_yes、vmail_plan_no 和 vmail_plan_yes。

```
# 为intl_plan和vmail_plan创建二值(虚拟)指标变量
# 并将它们与新数据框中的churn列连接起来
intl_dummies = pd.get_dummies(churn['intl_plan'], prefix='intl_plan')
vmail_dummies = pd.get_dummies(churn['vmail_plan'], prefix='vmail_plan')
churn_with_dummies = churn[['churn']].join([intl_dummies, vmail_dummies])
print(churn_with_dummies.head())
```

下面这段代码演示了如何将一列按照四分位数进行划分，为每个四分位数创建二值指标变量，并将新列添加到原来的数据框中。qcut 函数将 total_charges 列按照四分位数进行划分，并使用 qcut_names 中的名称对每个四分位数进行标记。get_dummies 函数为四分位数创建 4 个二值指标变量，并使用 total_charges 作为新列的前缀。最终结果为 4 个新的虚拟变量：total_charges_1st_quartile、total_charges_2nd_quartile、total_charges_3rd_quartile、total_charges_4th_quartile。join 函数将这 4 个变量追加到数据框 churn 中。

```
# 将total_charges按照分位数分组,为每个分位数分组创建一个二值指标变量
# 并将它们加入到churn数据框中
qcut_names = ['1st_quartile', '2nd_quartile', '3rd_quartile', '4th_quartile']
total_charges_quartiles = pd.qcut(churn.total_charges, 4, labels=qcut_names)
dummies = pd.get_dummies(total_charges_quartiles, prefix='total_charges')
churn_with_dummies = churn.join(dummies)
print(churn_with_dummies.head())
```

最后一段代码创建了 3 个数据透视表。第一行代码在对 total_charges 列按照流失情况和客户服务通话次数进行透视转换（或行列分组）之后，计算每组的均值。结果是一长列数值，表示每个流失情况和客户服务通话次数组合的平均总费用。第二行代码表示对结果重新格式化，使用流失情况作为行，客户服务通话次数作为列。最后，第三行代码使用客户服务通话次数作为行，流失情况作为列，演示了指定要计算的统计量、处理缺失值和是否显示边际值的方法：

```
# 创建透视表
print(churn.pivot_table(['total_charges'], index=['churn', 'custserv_calls']))
print(churn.pivot_table(['total_charges'], index=['churn'],\
columns=['custserv_calls']))
print(churn.pivot_table(['total_charges'], index=['custserv_calls'],\
columns=['churn'], aggfunc='mean', fill_value='NaN', margins=True ))
```

7.3.1 逻辑斯蒂回归

在这个数据集中，因变量是一个二值变量，表示客户是否已经流失并不再是公司客户。线性回归不适合这种情况，因为它可能会生成小于 0 或大于 1 的预测结果，这在概率上是没有意义的。因为因变量是一个二值变量，所以需要将预测值限制在 0 和 1 之间。逻辑斯蒂回归可以满足这个要求。

逻辑斯蒂回归模型如下所示。

- $\Pr(y_i = 1) = \text{logit}^{-1}(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$

对于 $i = 1, 2, \cdots, n$ 个观测和 p 个输入变量

等价于：

- $\Pr(y_i = 1) = p_i$
- $\text{logit}(p_i) = (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})$

逻辑斯蒂回归通过使用逻辑函数（或称逻辑斯蒂函数）的反函数估计概率的方式来测量自变量和二值型因变量之间的关系。这个函数可以将连续值转换为 0 和 1 之间的值，这是个必要条件，因为预测值表示概率，而概率必须在 0 和 1 之间。这样，逻辑斯蒂回归预测的就是某种结果的概率，比如客户流失概率。

逻辑斯蒂回归通过一种能够实现极大似然估计的迭代算法来估计未知的 β 参数值。

逻辑斯蒂回归的语法与线性回归有一点区别。对于逻辑斯蒂回归，需要分别设置因变量和自变量，而不是将它们写在一个公式中：

```
dependent_variable = churn['churn01']
independent_variables = churn[['account_length', 'custserv_calls',\
'total_charges']]
independent_variables_with_constant = sm.add_constant(independent_variables,\
prepend=True)

logit_model = sm.Logit(dependent_variable, independent_variables_with_constant)\
.fit()

print(logit_model.summary())
print("\nQuantities you can extract from the result:\n%s" % dir(logit_model))
print("\nCoefficients:\n%s" % logit_model.params)
print("\nCoefficient Std Errors:\n%s" % logit_model.bse)
```

第一行代码创建一个变量 `dependent_varible` 并赋给它 `churn01` 列中的一系列值。

同样，第二行代码设定了用作自变量的 3 列，并将它们赋给变量 `independent_variables`。

然后，我们使用 `statsmodels` 的 `add_constant` 函数向输入变量中加入一列 1。

下一行代码拟合逻辑斯蒂模型，并将拟合结果赋给变量 `logit_model`。

最后 4 行代码向屏幕上打印出模型中具体的结果。第一行代码向屏幕上打印模型的摘要信息。这个摘要信息非常重要，因为其中包含了模型系数、系数标准差和置信区间、伪 R 方等模型详细信息。

下一行代码打印出一个列表，其中包含从模型对象 `logit_model` 中提取出的所有数值信息。检查了这个列表之后，提取出模型系数和它们的标准差。

接下来的 2 行代码提取出了这些值。`logit_model.params` 以一个序列的形式返回模型系数，这样便可以通过定位或名称提取出单个模型系数。同样，`logit_model.bse` 以一个序列的形式返回系数标准差。输出结果如图 7-7 所示。

```
Optimization terminated successfully.
         Current function value: 0.363480
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                churn01   No. Observations:                 3333
Model:                          Logit   Df Residuals:                     3329
Method:                           MLE   Df Model:                            3
Date:                Sat, 28 May 2016   Pseudo R-squ.:                  0.1216
Time:                        13:20:39   Log-Likelihood:                -1211.5
converged:                       True   LL-Null:                       -1379.1
                                        LLR p-value:                 2.234e-72
==================================================================================
                     coef    std err          z      P>|z|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------
const             -7.2205      0.394    -18.309      0.000        -7.993    -6.448
account_length     0.0012      0.001      0.927      0.354        -0.001     0.004
custserv_calls     0.4443      0.037     12.129      0.000         0.373     0.516
total_charges      0.0729      0.005     13.448      0.000         0.062     0.084
==================================================================================
```

图 7-7：使用 3 个变量对客户流失进行多元逻辑斯蒂回归

7.3.2 系数解释

对逻辑斯蒂回归系数的解释不像线性回归那么直观，因为逻辑斯蒂函数的反函数是条曲线，这说明自变量一个单位的变化所造成的因变量的变化不是一个常数。

因为逻辑斯蒂函数的反函数是一条曲线，所以必须选择使用哪个函数值来评价自变量对成功概率的影响。和线性回归一样，截距系数的意义是当所有自变量为 0 时成功的概率。有些时候 0 是没有意义的，所以另外一种方式是当自变量都取均值时，看看函数的值有何意义：

```python
def inverse_logit(model_value):
    from math import exp
    return (1.0 / (1.0 + exp(-model_value)))

at_means = float(logit_model.params[0]) + \
    float(logit_model.params[1])*float(churn['account_length'].mean()) + \
    float(logit_model.params[2])*float(churn['custserv_calls'].mean()) + \
    float(logit_model.params[3])*float(churn['total_charges'].mean())

print("Probability of churn at mean values: %.2f" % inverse_logit(at_means))
```

第一段代码定义了一个函数 `inverse_logit`，将线性模型得出的连续预测值转换为 0 和 1 之间的概率值。

第二段代码计算出当所有自变量取均值时观测的预测值。4 个 `logit_model.params[…]` 值是模型系数，3 个 `churn['…'].mean()` 分别是账户时间、客户服务通话次数和总费用的均值。给定了模型系数和各变量的均值，方程变为 −7.2+(0.001*101.1)+(0.444*1.6)+(0.073*59.5)，所以变量 `at_means` 的值为 −2.068。

最后一行代码在屏幕上打印出逻辑斯蒂函数的反函数在 `at_means` 处的值，保留两位小数。−2.068 处的反函数值为 0.112，所以当账户时间、客户服务通话次数和总费用均取均值时，客户流失的概率就是 11.2%。

同样，要计算某个自变量一个单位的变化造成的因变量的变化，可以通过计算当某个自变量从均值发生一个单位的变化时，成功概率发生了多大的变化。

例如，下面来计算一下当客户服务通话次数在均值的基础上发生一个单位的变化时，对客户流失概率造成的影响：

```
cust_serv_mean = float(logit_model.params[0]) + \
    float(logit_model.params[1])*float(churn['account_length'].mean()) + \
    float(logit_model.params[2])*float(churn['custserv_calls'].mean()) + \
    float(logit_model.params[3])*float(churn['total_charges'].mean())

cust_serv_mean_minus_one = float(logit_model.params[0]) + \
    float(logit_model.params[1])*float(churn['account_length'].mean()) + \
    float(logit_model.params[2])*float(churn['custserv_calls'].mean()-1.0) + \
    float(logit_model.params[3])*float(churn['total_charges'].mean())

print("Probability of churn when account length changes by 1: %.2f" % \
(inverse_logit(cust_serv_mean) - inverse_logit(cust_serv_mean_minus_one)))
```

第一段代码与计算 `at_means` 的代码相同。第二段代码也与其基本相同，区别在于将客户服务通话次数的均值减去了 1。

最后一行代码打印出了两个逻辑斯蒂函数反函数值的差，其中一个是所有自变量为均值时的反函数值，另一个是两个自变量为均值，客户服务通话次数为均值减 1 时的反函数值。

在这个示例中，`cust_serv_mean` 的值与 `at_means` 相同，都是 −2.068。`cust_serv_mean_minus_one` 的值是 −2.512。−2.068 的反函数值减去 −2.512 的反函数值的结果是 0.0372，所以在均值附近减少一次客户服务通话就对应着客户流失概率提高 3.7 个百分点。

7.3.3 预测

与前面的葡萄酒质量预测一样，你也可以使用这个拟合模型来对“新”观测进行预测：

```
# 在churn数据集中
# 使用前10个观测创建10个“新”观测
new_observations = churn.ix[churn.index.isin(range(10)),\
independent_variables.columns]
new_observations_with_constant = sm.add_constant(new_observations, prepend=True)

# 基于新观测的账户特性
# 预测客户流失可能性
y_predicted = logit_model.predict(new_observations_with_constant)

# 将预测结果保留两位小数并打印到屏幕上
y_predicted_rounded = [round(score, 2) for score in y_predicted]
print(y_predicted_rounded)
```

同样，变量 `y_predicted` 中包含着 10 个预测值。为了使输出更简单易懂，可以将预测值保留两位小数。现在我们得到了可用的预测值，如果使用的是真正的新观测，那么就可以使用这些预测值来评价模型了。

第 8 章

按计划自动运行脚本

本书到目前为止，介绍了很多基本技术。在讨论了 Python 基础知识之后，我们对文本文件、CSV 文件、Excel 文件和数据库中的数据进行了处理，并应用这些新知识解决了 3 种常见的商业分析问题。在这些示例中，命令行中的脚本都是通过手动运行的。就像这样：

```
python my_python_script.py input_file.txt output_file.csv
```

这是一种最常见的运行脚本的方法，也是完全可以接受的，但是，当你需要定期运行脚本时，应该怎么办呢？如果没有别的运行脚本的方法，那么就需要你时刻记住要在某个时间使用命令行运行脚本。显然，这不是定期运行脚本的最优方法。在这种情况下，就需要另外一种方法，来按计划定期地运行脚本。

在 Windows 系统和 macOS 系统中，都有可以定期运行脚本和其他可执行文件的程序。微软称这个程序为 Task Scheduler（任务计划程序）；在 Unix 系统和 macOS 系统中，这样的程序称为 cron（定时任务，你可能听说过 crontab files 或 cron jobs）。本书的重点在于如何在 Windows 系统中运行脚本，所以下一节将演示在 Windows 系统中使用任务计划程序安排脚本定期运行的方法。同时，你也应该了解一下如何在 macOS 系统和 Unix 系统中安排定时任务，所以在下面的内容中还会演示在这两种操作系统中使用 cron 来安排 Python 脚本定期运行的方法。

8.1　任务计划程序（Windows系统）

为了演示在 Windows 系统中使用任务计划程序安排脚本定期运行的方法，首先需要选择一个 Python 脚本。为简单起见，可以使用第 5 章中最后一个应用程序的脚本 3parse_text_file.py。在这个应用程序中，用前面提到的脚本来解析 MySQL 错误日志文件。这个应用程序非常适合我们的要求，因为错误日志正是需要定期分析的一种文件。例如，你可能需要每天、每

周或每月分析一次数据库错误日志，来弄清楚某种错误发生的频率，以便有针对性地进行维护和修复工作。最后，尽管这个示例演示的是定期运行 Python 脚本的方法，但请注意，你完全可以使用任务计划程序安排其他类型的脚本和可执行文件定期运行。

首先，确认一下你在第 5 章最后一个应用程序中创建的两个文件（也就是 3parse_text_file.py 和 mysql_server_error_log.txt）都保存在桌面上了。如果你已经将这两个文件保存在了桌面上，那么下面的指导步骤和屏幕截图中的路径就非常容易理解了。当然，你可以将文件保存在另一个位置，然后在任务计划程序中修改文件路径，使它们指向你在计算机中保存文件的地方。

要打开任务计划程序，先单击开始按钮，找到控制面板→系统和安全→管理工具，然后双击任务计划程序（参见图 8-1）。如果系统提示需要输入管理员密码或进行确认，就输入管理员密码或进行确认。

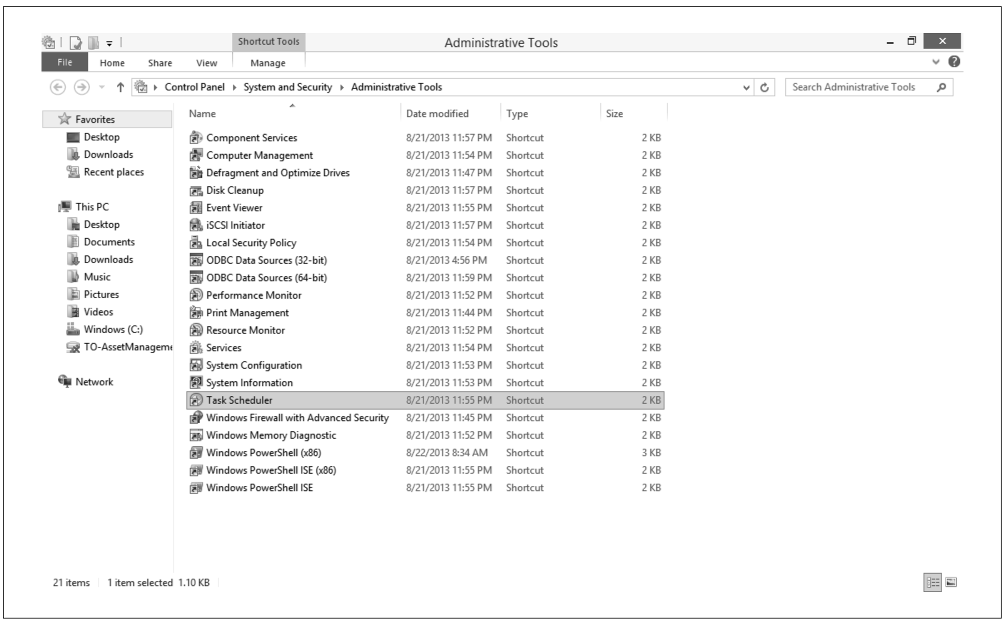


图 8-1：管理工具窗口中高亮显示的任务计划程序

请注意屏幕上方的文件路径：控制面板→系统和安全→管理工具。在管理工具列表中，任务计划程序在蓝色矩形背景下被高亮显示出来。

双击鼠标之后，任务计划程序将被打开。任务计划程序打开之后，屏幕如图 8-2 所示。

请注意右上角的可用操作列表（例如：连接到另一台计算机、创建基本任务等）。这些操作同样可以使用左上角的“操作”菜单来进行。

要安排一项任务，点击左上角的“操作”菜单，然后点击“创建基本任务”，也可以双击右上角的“创建基本任务”。通过任何一种操作，都可以打开“创建基本任务向导”。

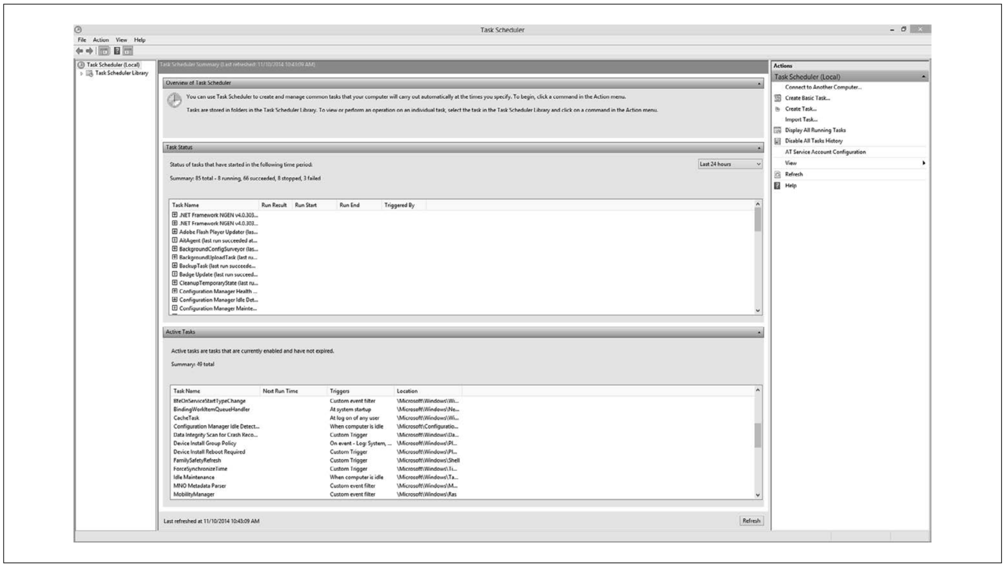


图 8-2：任务计划程序打开之后的初始界面

通过填写向导主窗口的名称与描述域，可以命名并描述你要创建的任务（参见图 8-3）。因为你要创建一个任务来运行 Python 脚本，去定期解析错误日志文件，所以此处将任务命名为“Parse Error Log File”，并描述为：“This task schedules a Python script, 3parse_text_file.py, to parse an error log file on a monthly basis.”。填写完名称与描述域后，点击“下一步”。

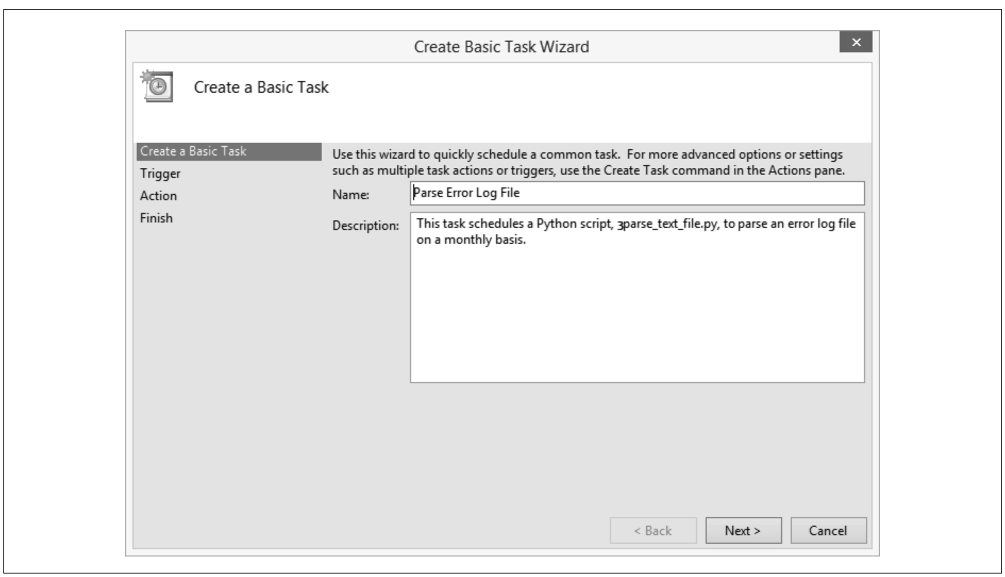


图 8-3：创建基本任务的界面，用来命名与描述要安排的任务

点击了“下一步”之后，任务向导会转到“触发器”标签页（参见图 8-4）。在“触发器”

标签页中，你可以选择任务开始的时间。因为需要每月运行一次脚本，所以选择“每月”单选按钮，然后点击“下一步”。

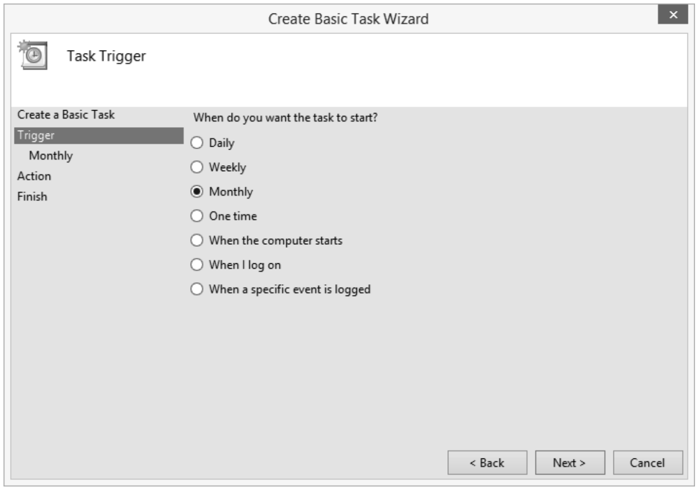


图 8-4：任务触发器界面，用来设置任务开始的时间

点击了“下一步”之后，任务向导会转到“每月”标签页（参见图 8-5）。在“每月”标签页中，你可以设置任务开始的时间。因为需要每月运行一次脚本，所以可以选择当前月份最后一天的上午 9:00 作为开始时间。选择“跨时区同步”复选框，并选择一年中的所有月份（一月、二月、三月……）和每月的最后一天。完成这些选择之后，点击“下一步”。

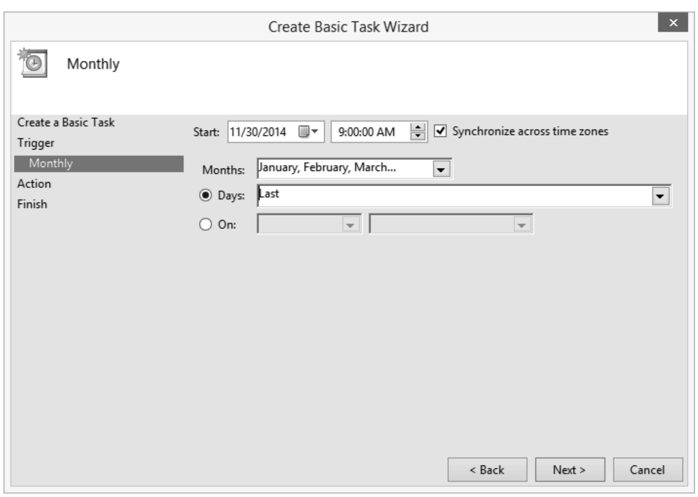


图 8-5：“每月”界面，用来设置任务运行的时间

点击了“下一步”之后，任务向导会转到“操作”标签页（参见图 8-6）。在“操作”标签页中，你可以选择想让任务执行的操作。因为需要任务去运行一个 Python 脚本，所以选择“启动程序”单选按钮，然后点击“下一步”。

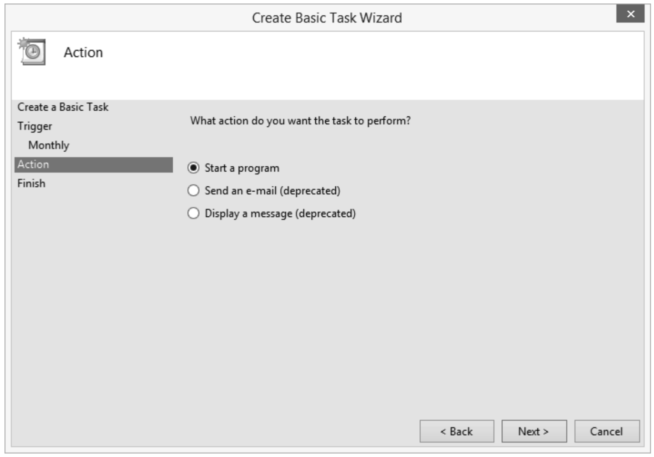


图 8-6：“操作”界面，用来设置任务执行的操作

点击了“下一步”之后，任务向导会转到“启动程序”标签页（参见图 8-7）。在“启动程序”标签页中，你可以设置要启动的程序或脚本。使用“浏览”按钮可以找到桌面上的 3parse_text_file.py。另外，这个脚本还需要两个命令行参数：输入文件名 mysql_server_error_log.txt 和输出文件名 mysql_errors_count.csv，可以在“添加参数（可选）”文本框中填写这两个参数。输入了 Python 脚本路径名和输入输出文件名之后，点击“下一步”。

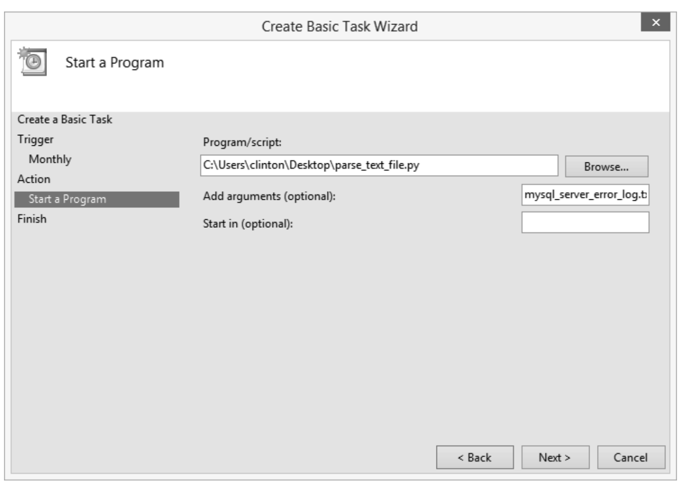


图 8-7：“启动程序”界面，用来设置任务运行的程序或脚本，以及程序或脚本需要的命令行参数

点击了“下一步”之后，任务向导会转到“完成”标签页（参见图 8-8）。“完成”标签页显示了你在任务向导中输入的所有信息的摘要，这样你可以在完成任务计划之前检查一下信息是否正确。检查项目包括名称、描述、触发器和操作域中的信息，确认它们是正确的。当核实了所有信息都正确之后，点击“完成”。

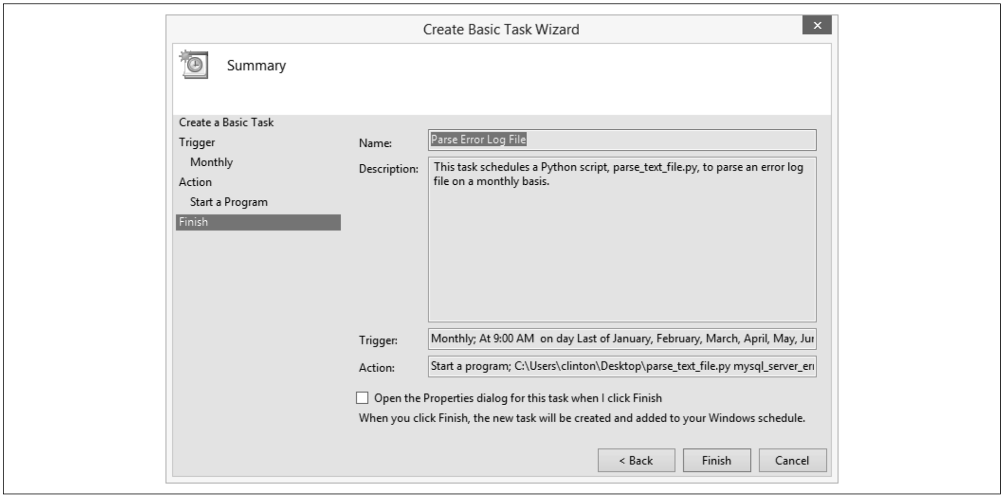


图 8-8：摘要界面，显示所有输入信息，用来确认任务被设置为执行你需要的操作

点击“完成”之后，任务向导会将你的任务添加到任务计划程序库中，并返回任务计划程序主界面（参见图 8-9）。要想查看一下新建的任务计划，可以点击主界面左上角的任务计划程序库。这时，在中上窗格中会显示你新建的任务，它可能位于其他任务之间。如果你点击了中上窗格中新任务的名称，将会在中下窗格中看到任务相关信息的摘要标签页（例如：常规、触发器、操作等）。最后，如果你想编辑或删除任务，可以先在中上窗格中点击选择任务，然后分别点击主界面右上角的“属性”或“删除”。

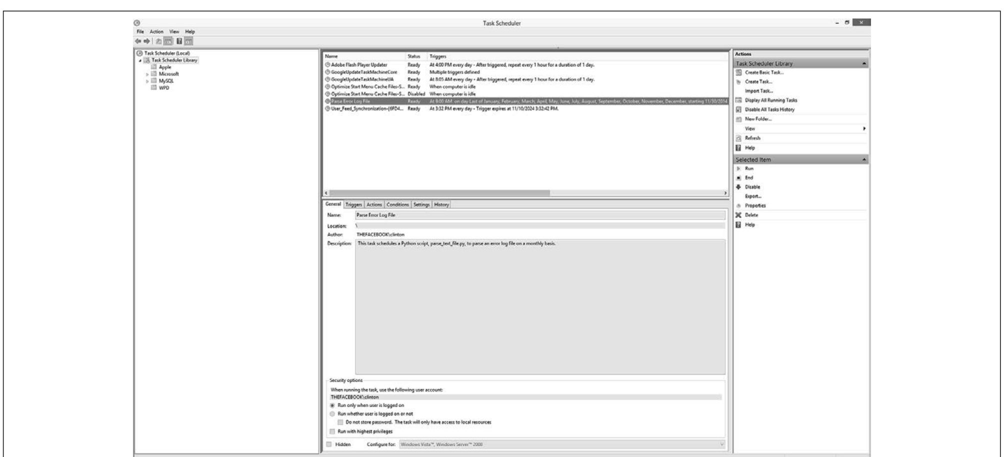


图 8-9：任务计划程序库界面，用来创建、查看、编辑和删除计划好的任务

通过按计划自动地定期运行 Python 脚本和其他可执行文件，你可以消除忘记手动运行脚本的可能。此外，相对于手动运行脚本，你可以通过自动运行脚本极大地提高工作效率。当你的业务对数据处理与分析脚本越来越依赖时，手动运行脚本的方式会更加不可行。

8.2 cron工具（macOS系统和Unix系统）

如你所见，Windows 系统中提供了任务计划程序，用来使脚本和其他可执行文件按计划自动定期运行。在 macOS 系统和 Unix 系统中，与之相似的程序称为 cron。

cron 程序依赖于 cron 表文件和 cron 任务来确定何时运行特定的可执行文件。cron 表文件是一个纯文本文件，你可以创建这个文件，在文件中列出所有想按计划自动运行的可执行文件，以及每个文件开始运行的具体时间。一个 cron 任务就是 cron 表文件中的一行，设定了一个要运行的可执行文件（例如：3parse_text_file.py）以及这个文件开始运行的时间（例如：每月运行）。

cron 表文件中的 cron 任务的具体语法一开始理解起来会有些困难。每行中的前 5 项设置了运行可执行文件的频率。这 5 项从左到右分别是：分钟（0~59）、小时（0~23）、天（1~31）、月（1~12）和星期几（0~6，星期天为 0）。每行中的最后一项设置了要按照设定频率运行的可执行文件。

有若干种方法可以设定前 5 项中的值。如果你想让可执行文件在某个项目的所有可能取值时都可以运行，那么就将这个项目设为一个星号（*）。例如，如果你想让文件每天都运行，那么就将第 3 个项目设为星号。相反，如果你想让文件在一个具体的时间运行，那么就应该将前两项设置为具体数值。例如，如果你想让文件在下午 3:10 开始运行，那么就应该将第一项设为 10，第二项设为 15（下午 3 时 =15 时）。

理解如何设置 cron 任务的最好方法是看几个例子，下面的例子给出了 cron 表文件中的 3 个 cron 任务：

```
10 15 * * * /Users/clinton/Desktop/analyze_orders.py
0 6,12,18 * * 1-5 /Users/clinton/Desktop/update_database.py
30 20 * * 6 /Users/clinton/Desktop/delete_temp_files.sh
```

第一行设置了 analyze_orders.py 应该在每月每日的下午 3:10 运行。第二行设置了 update_database.py 应该在每月的每个工作日（星期一至星期五）的早上 6:00、中午 12:00 和傍晚 6:00 各运行一次。第三行设置了 delete_temp_files.sh（一个 Bash 脚本）应该在每月的每个星期六的晚上 8:30 分运行。

这 3 个例子给出的是一些常用的 cron 任务设置。但是，你可能需要以一个特殊的频率运行脚本。例如，你可能需要在每月的第一个星期一运行脚本。在你已经确定了运行脚本的频率，但还不确定如何设置 cron 任务的时候，可以在互联网上搜索具体的语法（有人已经为你提供了解决方案）。例如，使用“cron job first Monday of month”进行一次快速搜索，就能够找到下面的语法形式，可以在每月的第一个星期一的上午 11:00 运行 Python 脚本 every_first_Monday.py：

```
00 11 1-7 * * [ "$(date '+\%a')" = "Mon" ] &&\
/Users/clinton/every_first_monday.py
```

8.2.1 cron表文件：一次性设置

我们已经从概念上理解了 cron 表文件和 cron 任务，下面就创建一个 cron 表文件，并设置一个 cron 任务，定期运行 Python 脚本 3parse_text_file.py。

cron 表文件的创建本质上是一种一次性设置。创建了 cron 表文件之后，就不再需要重新创建了。你可以在已有的 cron 表文件之中添加、修改或删除 cron 任务，来管理想定期自动运行的可执行文件。

要创建一个新的空 cron 表文件，打开终端窗口，输入以下命令：

```
touch crontab_file.txt
```

要想加载 cron 表文件（就是让操作系统加载 cron 表文件并按计划执行其中的命令），在命令行中输入以下命令，然后按回车键：

```
crontab crontab_file.txt
```

最后，从创建 crontab_file.txt 的位置删除它。要完成这个操作，在命令行中输入以下命令，然后按回车键：

```
rm crontab_file.txt
```

这样就完成了一个空的 cron 表文件的创建，一次性设置完成。图 8-10 中的屏幕截图展示了上面的 3 种一次性设置命令，还有用来编辑 cron 表文件的 `crontab -e` 命令。

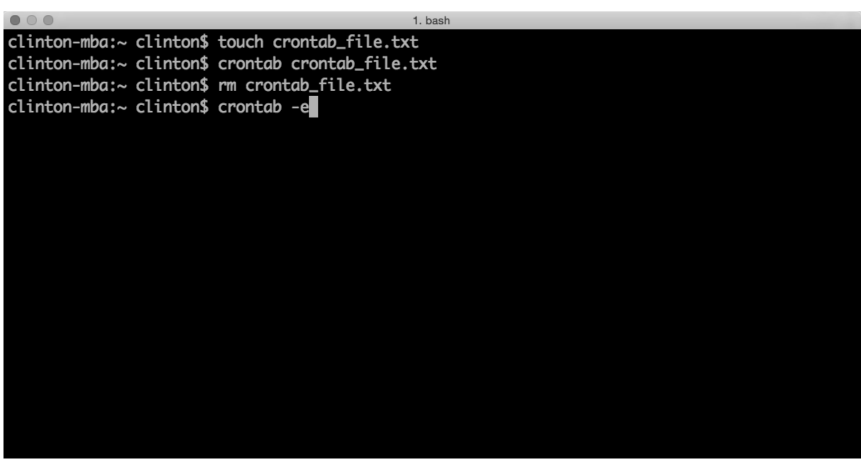


图 8-10：图中的 3 条命令可以在命令行中使用，建立一个空的 cron 表文件。第四条命令 crontab -e 打开新建的 cron 表文件进行编辑

8.2.2 向cron表文件中添加cron任务

现在向 cron 表文件中添加一项 cron 任务。首先打开一个 cron 表文件进行编辑，输入以下命令，然后按回车键：

```
crontab -e
```

当你执行了 `crontab -e` 命令后，cron 表文件会在基于 Unix 的文本编辑器中打开，这样的文本编辑器包括 Nano、vi/Vim 或 Emacs。你可以在当前行中输入 cron 任务命令，用回车键将光标移到下一个空行，然后使用适当的关键字序列（下面会解释）来保存修改和退出文件。

有一种特殊情况，如果是用 vi/Vim 打开了文件，那么你使用的编辑器就有两种模式：命令模式和插入模式。文件刚打开时，编辑器处于命令模式，此时你输入的按键为作用在文件上的命令，而不是要输入文件中的文本。要想从命令模式切换到插入模式（插入模式允许你向文件中输入文本），需要输入命令 `i`。进入插入模式之后，你就可以在当前行中输入 cron 任务命令，用回车键将光标移到下一个空行，然后使用适当的关键字序列来保存修改和退出文件。

打开 cron 表文件，在当前行中输入以下命令，然后按回车键将光标移至下一个空行（参见图 8-11）：

```
00 09 28-31 * * [ "$(date -v+1d '+\%d')" = "01" ] &&\
/Users/clinton/Desktop/3parse_text_file.py
```

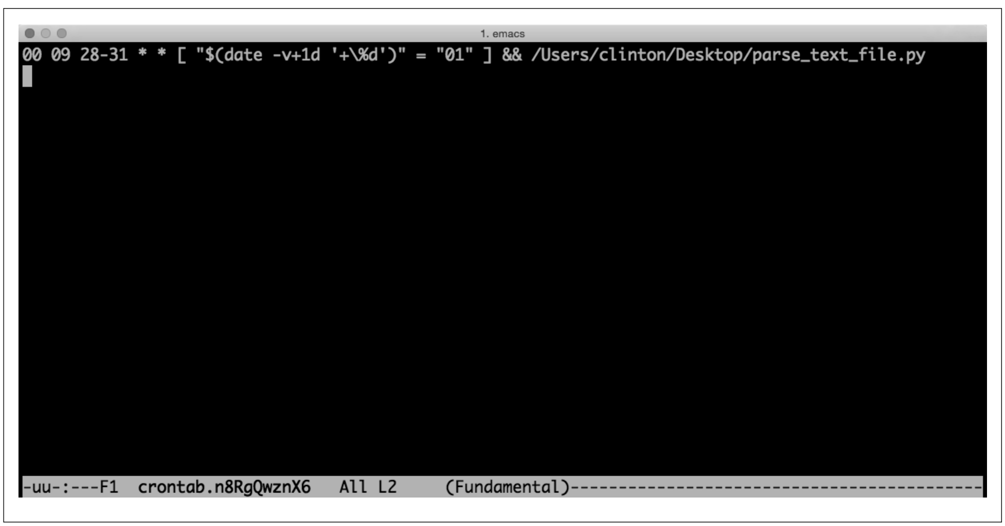


图 8-11：输入 cron 表文件中的命令，在每月最后一天上午 9:00 运行脚本 3parse_text_file.py

这条命令与之前在 Windows 系统的任务计划程序中设置的参数是一致的，都是在每月最后一天的上午 9:00 运行脚本。命令左侧的 5 项表示这个任务应该在满足方括号中条件语句的前提下，在每月的 28、29、30、31 日中某一天的上午 9:00 运行。方括号中的语句检验的是，如果将当前日期加 1，那么所得的日期是否是 01（即下个月的第一天）。这条语句确保了脚本在每月的最后一天运行，不论这一天是 2 月 28 日、6 月 30 日还是 10 月 31 日。cron 程序会检查这些频率参数和语句，如果语句为真，cron 任务就会运行 3parse_text_file.py。每月最后一天的上午 9:00，都会运行这个脚本。

请注意光标（一个白色小长方形）在你输入的命令的下一个空行内。cron 表文件中可以有多个 cron 任务，每个任务占一行，但是你必须在最后一个 cron 任务后面按回车键，以使 cron 任务行结束，并使光标位于文件中的最后一个空行内。

既然你已经在 cron 表文件中输入了一个 cron 任务命令，现在就需要保存对文件的修改并退出文件了。根据你使用的编辑器，输入下列命令序列中的一个来保存修改并退出 cron 表文件：

- Nano：Ctrl+o，Ctrl+x
- vi/Vim：，`w`，`q`
- Emacs：Ctrl+x，Ctrl+s，Ctrl+x，Ctrl+c

保存了修改并且退出了 cron 表文件之后，查看一下 cron 表文件中的内容，看看保存在文件中的新建的 cron 任务。要查看 cron 表文件中的内容，输入以下命令，然后按回车键（参见图 8-12）：

```
crontab -l
```

按下回车键之后，就可以看到 cron 表文件中的内容被打印到了屏幕上。如屏幕截图所示，cron 表文件中包含了我们的 cron 任务命令，在每月最后一天的上午 9:00 运行脚本 3parse_text_file.py。

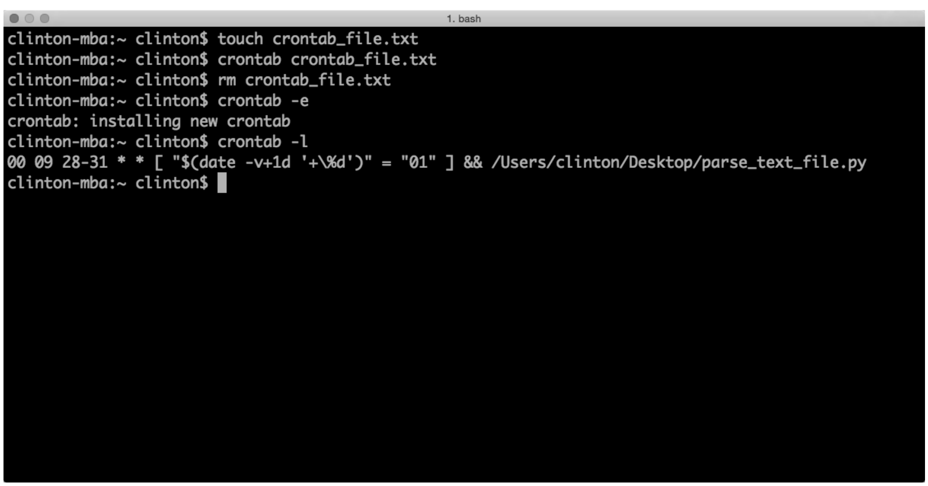


图 8-12：在终端窗口中使用 crontab -l 显示 cron 表文件中的内容

要编辑或删除一个 cron 任务，先输入 `crontal -e` 打开 cron 表文件。如果你想编辑 cron 任务，对你想修改的位于某行的 cron 任务进行修改即可。如果你想删除 cron 任务，删除包含这个任务的行即可。对于以上两种情况，都要确认光标停留在文件中的最后一个空行内。然后，根据你使用的文本编辑器输入适当的按键序列，保存修改并退出 cron 表文件。

尽管你在日常工作中使用 Windows，但是知道如何计划 cron 任务也是非常重要的。因为有些时候，你可能会需要计划一个 cron 任务，所以知道如何在不同的操作系统下实现自动工作是非常有用处的。

本章篇幅比其他各章都短，但却是本书的一个重要补充，因为其内容可以使你定期自动运行所需脚本。其他各章教会了你扩展数据处理和分析能力的技术，本章则在规模化和自动化方面增强了这些技术。通过自动运行所需脚本，你可以减少忘记运行脚本的概率，并可以使用节省下来的时间去进行更重要的工作。

第 9 章

从这里启航

这是本书的最后一章，到目前为止，书中已经介绍了很多知识和技术，包括 Python 基础知识以及分析任意数量的文本文件、CSV 文件、Excel 文件和数据库中数据的方法。我们已经掌握了如何从这些数据源中选择特定的行与列，如何聚合数据并计算基本统计量，以及如何将结果写入输出文件。我们完成了 3 个常见的商业分析应用，这些应用要求我们创造性地且有效率地使用学过的知识和技术。我们还学习了如何通过一些扩展模块创建最常用的统计图表，以及如何通过 StatsModels 包来估计回归模型和分类模型。最后，我们学习了如何按计划自动定期运行脚本，这样便可以节省出时间来进行其他更重要的分析工作。如果你一直跟随着本书中的示例进行学习和实践，那么能否体会到，你正经历着从门外汉到编程高手的转变？

到目前为止，你可能会非常想知道下一步应该做些什么。也就是说，在掌握了使用 Python 规模化和自动化地完成数据分析任务之后，还应该学习些什么？本章将会介绍标准 Python 发布版本中一些其他的功能，在你刚开始学习 Python 时，这些功能不是必要的，但它们确实是非常有趣而且实用的。在学习了本书前面的章节之后，希望你会发现，这些功能更容易理解，而且能更方便地扩展前面已经学习过的技术。

本章还会讨论一下 NumPy、SciPy 和 Scikit-Learn 扩展包，因为它们分别提供了基础的数据容器和向量运算、可以用于科学计算和统计分析的数学分布和检验，以及统计建模方法和机器学习功能，pandas 包依赖于这些功能，StatsModels 包则在这些功能的基础上进行了扩展。例如，Scikit-Learn 提供了数据预处理，数据降维，回归、分类与聚集模型估计，模型比较与选择，交叉验证等强大的功能。这些方法可以帮助你创建、检验和选择模型，这样得出的模型对新数据是具有鲁棒性的，使用这种模型和新数据更有可能做出准确的预测。

最后，本章还要再介绍几种数据结构，它们有助于你更加熟练地使用 Python。本书重点介绍的数据结构包括列表、元组和字典，因为它们是功能强大的基础数据容器，完全可以满

足初级编程的需要（其实，对于你现在的水平，只学习这 3 种数据结构也足够了）。但是，还有一些其他数据结构，像栈、队列、堆、树、图，等等，它们对一些特殊的要求更加适用。

9.1 更多的标准库模块和内置函数

对于很多 Python 内置模块和标准库模块，以及用于读写和分析文本文件、CSV 文件、Excel 文件和数据库中数据的函数，相信大家已经很熟悉了。例如，本书前面章节已经使用过 Python 内置的 `csv`、`datetime`、`re`、`string` 和 `sys` 模块，以及一些 Python 内置函数，如 `float`、`len` 和 `sum`。

但是，相对于 Python 标准库中的所有模块和函数来说，前面用到的只是冰山一角。所以，本章还要介绍几种模块和函数，因为它们对于数据处理与分析特别重要。这些模块和函数在前面的章节中没有涉及，要么因为它们不适用于那些特定的示例，要么因为它们是更高级的选择。但你应该知道，在特定的数据分析任务中，这些模块和函数是可以发挥作用的。所以如果你想给自己设定一个挑战，那么就应该努力做到每天或每两天掌握一种以下的新技术。

9.1.1 Python标准库（PSL）：更多的标准模块

- `collections`（PSL8.3）
  这个模块实现了一些特殊的数据容器类型，可以作为 Python 内置数据容器 `dict`、`list`、`set` 和 `tuple` 的替代品。一些常用于数据分析的容器是 `deque`、`Counter`、`defaultdict` 和 `OrderedDict`。
- `random`（PSL9.3）
  这个模块实现了可用于各种分布的伪随机数生成器。功能包括从一个取值范围内随机选择一个整数、从序列中随机选择一个元素、随机重排一个序列、无放回随机抽样，以及从均匀分布、正态（高斯）分布、伽玛分布、贝塔分布和其他分布中随机选择值。
- `statistics`（PSL9.7）
  这个模块提供了为数值型数据计算常用统计量的功能。它可以计算衡量数据集中程度的统计量，比如均值、中位数和众数，还可以计算衡量数据分散程度的统计量，比如方差和标准差。
- `itertools`（PSL10.1）
  这个模块为一些常用的数据算法提供一系列存储高效、运算快速的标准化迭代器（也称生成器）。这些迭代器可用于序列合并与分离、输入数据转换、新数据生成以及数据筛选和分组。
- `operator`（PSL10.3）
  这个模块提供了对应于 Python 内置运算符的一组高效函数。这些函数中既包括可以执行对象比较、逻辑运算、数学运算和序列运算的函数，也包括可以实现属性扩展和项目查找的函数。

这 5 个标准模块只是 Python 标准库中所有模块的一小部分。实际上，Python 标准库中有超出 35 个分类，每个分类中都提供了关于某个专题的各种各样的模块和函数。因为这些模块都是内置于 Python 中的，所以你可以用 `import` 语句直接使用它们，例如：`from itertools import accumulate` 或者 `from statistics import mode`。要了解更多关于 Python 标准模块的信息，可以参考 Python 标准库（https://docs.python.org/3/library/index.html）。

9.1.2 内置函数

同上面刚讨论过的标准模块一样，有些内置函数在以前的章节中没有涉及，但是它们对于数据处理和分析非常重要。和模块一样，在特定的数据分析任务中，这些函数是可以发挥作用的。将下面这些函数加入你的 Python 工具箱是非常有用处的。

- `enumerate()`
  将一个序列扩展为一个元组 (`index`, `value`) 列表。
- `filter()`
  在一个序列上应用一个函数，返回使函数为真的序列中的值。
- `zip()`
  将两个序列中相同位置的值合并为一个元组，从而使两个序列合并成一个序列。

这些函数是内置在 Python 中的，所以你可以直接使用它们。要了解更多关于 Python 内置函数的信息，可以参考 Python 标准库（https://docs.python.org/3/library/functions.html）。此外，看一下别人是如何使用这些函数完成特定分析任务的对你也很有帮助。如果你想学习一下其他人的方法，可以使用 Google 或 Bing 搜索“python enumerate examples”或者“python zip examples”，这样可以检索出一大批有用的示例。

9.2 Python包索引（PyPI）：更多的扩展模块

正如前面所见，标准 Python 安装版本中带有大量内置功能，其中的模块可以存取文本文件和 CSV 文件、处理文本和数值型数据、计算统计量，还可以实现本书中未曾提及的其他大量强大功能。

但是，本书还使用了一些扩展模块，比如 `xlrd`、`matplotlib`、`MySQL-python`、`pandas` 和 `statsmodels`，这些模块提供了 Python 标准库中不具备的功能。实际上，有若干种侧重于数据处理的重要扩展模块，在下载安装之后，它们可以提供一些强大的功能，包括数据可视化、数据操作、统计建模和机器学习。这些扩展模块包括 NumPy、SciPy、Scikit-Learn、xarray（原来称作 xray）、SKLL、NetworkX、PyMC、NLTK、Cython。

这些模块以及其他扩展模块，都可以从 Python 包索引网站（https://pypi.python.org/pypi）上下载。此外，对于需要区别 32 位系统和 64 位系统的 Windows 用户来说，可以在 Unofficial Windows Binaries for Python Extension Packages 网站（http://www.lfd.uci.edu/ ~ gohlke/pythonlibs/ ）上找到大量扩展包的 32 位版本和 64 位版本。

9.2.1 NumPy

NumPy（读作“Num Pie”）是一个基础 Python 扩展包，它（主要）为数值型数据提供了一个快速高效的多维数据容器 `ndarray`。它还提供了向量化版本的数学和统计函数，使你可以不使用 `for` 循环来操作数组。NumPy 提供的强大功能可以对结构化数据（特别是数值型数据）进行读取、重塑、聚合、切片和切块。

pandas 是基于 NumPy 开发的，与 pandas 一样，NumPy 也封装和简化了很多你在本书中已经学习过的技术。NumPy 是一个基础的扩展包，它提供了强大的 `ndarray` 数据结构，可以进行向量运算，很多扩展包都是基于 NumPy 开发的。下面看一下 NumPy 中的主要功能。

1. 读写CSV文件和Excel文件

本书在第 2 章中讨论过如何使用内置的 `csv` 模块来读写 CSV 文件。为了要读取一个 CSV 文件，可以使用 `with` 语句打开输入文件，使用 `filereader` 对象读出文件中的所有行。同样，为了写入 CSV 文件，可以使用 `with` 语句打开输出文件，使用 `filewriter` 对象写入输出文件。在这两种情况下，都是使用 `for` 循环在输入文件的所有行中进行迭代，并对行做出相应的处理。

NumPy 将读写 CSV 文件和文本文件的过程简化为 3 个函数：`loadtxt`、`genfromtxt` 和 `savetxt`。默认情况下，`loadtxt` 函数假设输入文件的数据是浮点数，并由一定数量的空白字符隔开。当然，你可以通过函数中的附加参数修改这些默认值。

2. loadtxt

与第 2 章中读取文件的代码不同，如果你的数据集中没有标题行，而且数据是由空格隔开的浮点数，那么你可以通过下列语句将数据加载到 NumPy 数组，然后直接使用数据：

```
from numpy import loadtxt
my_ndarray = loadtxt('input_file.csv')
print(my_ndarray)
```

然后，你可以像本书之前讨论过的那样，对数据进行各种操作。再看看另一个示例，假设你有一个文件 people.txt，其中包含以下数据：

```
name    age    color    score
clint    32    green     15.6
john     30    blue      22.3
rachel   27    red       31.4
```

请注意这个数据集中包含一个标题行，而且列中的数据不全是浮点数。在这种情况下，你可以通过 `skiprows` 参数来跳过标题行，还可以为每列数据指定数据类型：

```
from numpy import dtype, loadtxt
person_dtype = dtype([('name', 'S10'), ('age', int), ('color', 'S6'),\
('score', float)])
people = loadtxt('people.txt', skiprows=1, dtype=person_dtype)
print(people)
```

通过创建一个结构化数组 `person_dtype`，可以指定姓名列中的值为最大长度是 10 个字符的字符串，年龄列中的值为整数，颜色列中的值为字符串，分数列中的值为浮点数。

在这个示例中，列是由空格隔开的，如果数据是由逗号隔开的，你可以在 loadtxt 函数中使用 delimiter=',' 来指定列分隔符为逗号。

3. genfromtxt

genfromtxt 函数试图更进一步地简化你的工作量，它甚至可以自动确定列中的数据类型。和 loadtxt 一样，genfromtxt 函数也提供了附加参数，可以方便地将各种类型的文件和数据读取到结构化数组中。

例如，你可以使用 names 参数来指定是否存在标题行，还可以通过 converters 参数对从输入文件中读取的数据进行修改和格式化：

```
from numpy import genfromtxt
name_to_int = dict(rachel=1, john=2, clint=3)
color_to_int = dict(blue=1, green=2, red=3)
def convert_name(n):
    return name_to_int.get(n, -999)
def convert_color(c):
    return color_to_int.get(c, -999)
data = genfromtxt('people.txt', dtype=float, names=True, \
converters={0:convert_name, 2:convert_color})
print(data)
```

上面这个示例是想把姓名列和颜色列中的数据从字符串转换为浮点数。示例中对每一列都创建一个字典，为原来的字符串映射出一个数值。同时还定义了两个辅助函数，为每个姓名值和颜色值找出字典中对应的数值，如果姓名和颜色没有出现在字典中，则返回 −999。

在 genfromtxt 函数中，参数 dtype 表示结果数据集中所有的值均为浮点数，参数 names 表示 genfromtxt 应该将第一行作为列标题，参数 converters 设置了一个字典，将列中的值映射到一个函数，这个函数会对列中的值进行转换。

4. 转换为NumPy数组

除了使用 loadtxt 和 genfromtxt，你还可以使用基础 Python 将数据读入一个列表，这个列表中的元素是列表或元组，另外也可以使用 pandas 将数据读入数据框，然后将这些对象转换为 NumPy 数组。

5. CSV文件

例如，假设你有一个 CSV 文件 myCSVInputFile.csv，其中包含以下数据：

```
2.1,3.2,4.3
3.2,4.3,5.2
4.3,2.6,1.5
```

你可以使用本书中讨论过的技术将这些数据读入一个列表的列表，然后将列表转换为 NumPy 数组：

```
import csv
from numpy import array
file = open('myCSVInputFile.csv', 'r')
file_reader = csv.reader(file)
data = []
for row_list in file_reader:
```

```
        row_list_floats = [float(value) for value in row_list]
        data.append(row_list_floats)
    file.close()
    data = array(data)
    print(data)
```

6. Excel文件

同样，如果你有一个 Excel 文件，可以使用 pandas 的 `read_excel` 函数将数据读入一个数据框，然后将数据框转换为 NumPy 数组：

```
from pandas import read_excel
from numpy import array
myDataFrame = read_excel('myExcelInputFile.xlsx')
data = array(myDataFrame)
print(data)
```

7. savetxt

NumPy 提供了 `savetxt` 函数，可以将数据写入 CSV 文件和其他类型的文本文件。首先，你需要设定输出文件名称，然后再设定要保存到文件中的数据即可：

```
from numpy import savetxt
savetxt('output_file.txt', data)
```

`savetxt` 默认使用科学计数形式保存数据。但你不想一直用这种方法，所以可以使用 `fmt` 参数来设置保存数据的形式。你还可以使用 `delimiter` 参数来设置列分隔符：

```
savetxt('output_file.txt', data, fmt='%d')
savetxt('output_file.csv', data, fmt='%.2f', delimiter=',')
```

还有，`savetxt` 默认不保存标题行。如果你想输出文件中包括标题行，那么可以使用 `header` 参数提供一个字符串。`savetxt` 默认在第一个列标题前面加上一个井号（`#`），以使这一行成为注释。你可以通过将 `comments` 参数设为空字符串来取消这个设置：

```
column_headings_list = ['var1', 'var2', 'var3']
header_string = ','.join(column_headings_list)
savetxt('output_file.csv', data, fmt='%.2f', delimiter=',', \
comments='', header=header_string)
```

8. 筛选行

如果你创建了一个 NumPy 结构化数组，那么就可以和 pandas 一样，使用筛选条件筛选出特定的行。例如，假设你创建了一个名为 `data` 的结构化数组，其中的列有成本、供应商代码、数量和交货时间，你可以使用下面的条件筛选出特定的行：

```
row_filter1 = (data['Cost'] > 110) & (data['Supplier'] == 3)
data[row_filter1]
row_filter2 = (data['Quantity'] > 55) | (data['Time to Delivery'] > 30)
data[row_filter2]
```

第一个筛选条件筛选出成本大于 110 **并且**供应商代码等于 3 的行。同样，第二个筛选条件筛选出数量大于 55 **或者**交货时间大于 30 的行。

9. 选取特定列

在结构化数组中选取一组特定的列有些难度，因为不同列的数据类型也是不同的。你可以定义一个辅助函数来查看选取出的列，然后根据列的数据类型做出相应的处理：

```
import numpy as np
def columns_view(arr, fields):
  dtype2 = np.dtype({name:arr.dtype.fields[name] for name in fields})
  return np.ndarray(arr.shape, dtype2, arr, 0, arr.strides)
```

然后你可以使用辅助函数查看从结构化数组中选取出的列。你还可以设置行筛选条件来筛选出特定的行，并同时选取特定的列，这与 pandas 中 `ix` 函数的用法是一样的：

```
supplies_view = columns_view(supplies, ['Supplier', 'Cost'])
print(supplies_view)
row_filter = supplies['Cost'] > 1000
supplies_row_column_filters = columns_view(supplies[row_filter],\
['Supplier', 'Cost'])
print(supplies_total_cost_gt_1000_two_columns)
```

10. 连接数据

NumPy 使用 `concatenate`、`vstack`、`r_`、`hstack` 和 `c_` 函数来简化多个数组的连接过程。`concatenate` 函数比其他函数更常用，它使用一个附加参数 `axis` 将一组数组连接起来，`axis` 表示数组应该垂直连接（`axis=0`）还是水平连接（`axis=1`）。`vstack` 函数和 `r_` 函数专门用来垂直连接数组，`hstack` 函数和 `c_` 函数专门用来水平连接数组。例如，下面是垂直连接数组的 3 种方式：

```
import numpy as np
from numpy import concatenate, vstack, r_
array_concat = np.concatenate([array1, array2], axis=0)
array_concat = np.vstack((array1, array2))
array_concat = np.r_[array1, array2]
```

这 3 个函数会得到同样的结果。在每行代码中，函数中的数组被垂直连接，一个在另一个的上面。如果你将结果赋给一个新的变量，那么就得到了一个更大的新数组，里面包含着两个输入数组中所有的数据。

同样，以下是平行连接数组的 3 种方式：

```
import numpy as np
from numpy import concatenate, hstack, c_
array_concat = np.concatenate([array1, array2], axis=1)
array_concat = np.hstack((array1, array2))
array_concat = np.c_[array1, array2]
```

这 3 个函数也得到了同样的结果。在每行代码中，函数中的数组被平行连接在一起，并排排列。

11. 其他功能

这一节介绍了 NumPy 的一些特点和功能，但是还有更多功能需要学习。NumPy 和基础 Python 的一个重要不同之处是 NumPy 可以进行向量化运算，就是说你可以对整个数组应用一个操作，这个操作会作用于数组中的所有元素，而不需要使用 `for` 循环。

例如，如果你有两个数组，array1 和 array2，你想把它们按照每个元素加在一起，那么只要简单地使用 array_sum=array1+array2 即可。这个操作会将两个数组中的所有元素分别相加，从而得到一个新的数组，新数组中每个位置上的值就是原来两个数组相同位置上的值的和。而且，向量化运算是使用 C 代码执行的，所以运算速度特别快。

NumPy 的另一个强大功能是可以在数组上执行统计计算。这些统计计算包括 sum、prod、amin、amax、mean、var、std、argmin 和 argmax。sum 和 prod 计算数组中所有值的总和与乘积。amin 和 amax 可以找出数组中的最小值和最大值。mean、var 和 std 计算数组中所有值的均值、方差和标准差。argmin 和 argmax 可以找出数组中最小值和最大值的索引位置。所有这些函数都具有 axis 参数，所以你可以设置 axis 参数，来指定是沿着列进行垂直计算（axis=0），还是沿着行进行水平计算（axis=1）。

要想下载 NumPy，或想了解更多关于 NumPy 的信息，请访问 NumPy 网站（http://www.numpy.org）。

9.2.2　SciPy

SciPy（读作“Sigh Pie”）是另一个 Python 基础扩展包，它提供用于科学计算与统计分析的各种分布和函数，以及数学、科研和工程方面的检验。SciPy 具有广泛的适用范围，它的功能分布在各种子扩展包内。重要的子扩展包如下。

- cluster
  提供聚集算法
- constants
  提供物理和数学常数
- interpolate
  提供用于插值和平滑样条的函数
- io
  提供输入 / 输出函数
- linalg
  提供线性代数运算
- sparse
  提供稀疏矩阵运算
- spatial
  提供空间数据结构和算法
- stats
  提供统计分布与函数
- weave
  提供 C/C++ 集成

从上面的列表中可以看出，SciPy 的子扩展包提供了非常丰富的操作和计算功能。例如，`linalg` 扩展包提供了在二维数组上进行快速线性代数运算的功能；`interpolate` 扩展包提供了在两点之间进行线性插值和曲线插值的功能；`stats` 扩展包提供了使用随机变量、计算并检验描述性统计量和回归分析的功能。

SciPy 是一个基础扩展包，除了提供丰富实用的数学和统计功能之外，它还是很多其他扩展包的基础，下面来看一下 SciPy 中的重要功能。

1. `linalg`

`linalg` 扩展包提供了所有基本线性代数运算的函数，包括矩阵求逆、求行列式、计算范数。它还包括了进行矩阵分解的函数，以及指数函数、对数函数和三角函数。另外，还有其他一些有用的函数可以快速求解线性方程组和线性最小二乘问题。

线性方程组。SciPy 提供了 `linalg.solve` 函数来计算线性方程组的解向量。假设我们想要求解以下的线性联立方程组：

- $x + 2y + 3z = 3$
- $2x + 3y + z = -10$
- $5x - y + 2z = 14$

可以使用一个系数矩阵、一个未知数向量和一个右侧向量来表示这个方程组。`linalg.solve` 函数使用系数矩阵和右侧向量可以求出未知数（就是 x、y 和 z）：

```
from numpy import array
from scipy import linalg
A = array([[1,2,3], [2,3,1], [5,-1,2]])
b = array([[3], [-10], [14]])
solution = linalg.solve(A, b)
print(solution)
```

方程组中 x、y 和 z 的值分别是：0.1667、−4.8333 和 4.1667。

最小二乘回归。SciPy 提供了 `linalg.lstsq` 函数来计算线性最小二乘问题的解向量。在计量经济学中会经常见到使用矩阵表示的最小二乘估计模型，如下所示：

- $y = Xb + e$

这里的 y 是一个表示因变量的向量，X 是自变量的系数矩阵，b 是要进行估计的解向量，e 是从数据中计算出的残差向量。`linalg.lstsq` 函数使用系数矩阵 X 和因变量 y 求解出解向量 b：

```
import numpy as np
from scipy import linalg
c1, c2 = 6.0, 3.0
i = np.r_[1:21]
xi = 0.1*i
yi = c1*np.exp(-xi) + c2*xi
zi = yi + 0.05 * np.max(yi) * np.random.randn(len(yi))
A = np.c_[np.exp(-xi)[:, np.newaxis], xi[:, np.newaxis]]
c, resid, rank, sigma = linalg.lstsq(A, zi)
print(c)
```

这里的 c1、c2、i 和 xi 仅用来构造因变量的初始形式 yi。下一行代码中构造的 zi 才是真正的因变量，它向 yi 中加入了一些随机扰动。lstsq 函数的返回值包括 c、残差（resid）、rank 和 sigma。c 中的两个值就是最小二乘问题的解，为 5.92 和 3.07。

2. interpolate

interpolate 扩展包提供了在已知数据点之间进行线性插值和曲线插值的功能。用于单变量数据的函数是 interp1d，用于多变量数据的函数是 griddata。这个扩展包还提供了样条插值函数和径向基函数，可以用来对数据进行平滑和插值。inter1d 函数接受两个数组为参数，返回一个函数对象，这个函数使用插值法找出新数据点的值：

```
from numpy import arange, exp
from scipy import interpolate
import matplotlib.pyplot as plt
x = arange(0, 20)
y = exp(-x/4.5)
interpolation_function = interpolate.interp1d(x, y)
new_x = arange(0, 19, 0.1)
new_y = interpolation_function(new_x)
plt.plot(x, y, 'o', new_x, new_y, '-')
plt.show()
```

在代码生成的图中，20 个蓝色圆点为初始数据点。在初始数据点之间，绿色直线连接着的新数据点就是插值。因为我没有在 interp1d 函数中设置 kind 参数，所以函数使用默认的线性插值法来计算插值。不过，你可以指定 quadratic、cubic 或另外的字符串或整数作为函数应该使用的插值方法。

3. stats

stats 扩展包提供的功能包括按照特定的分布求值、计算描述性统计量、统计检验、回归分析等。这个扩展包提供了超过 80 个连续随机变量和 10 个离散随机变量。它既可以进行单样本分析检验，也可以进行双样本比较检验。扩展包中还有进行核密度估计的函数，或通过一组数据对一个随机变量的概率密度函数进行估计。

描述型统计量。stats 扩展包提供了一些计算描述型统计量的函数：

```
from scipy.stats import norm, describe
x = norm.rvs(loc=5, scale=2, size=1000)
print(x.mean())
print(x.min())
print(x.max())
print(x.var())
print(x.std())
x_nobs, (x_min, x_max), x_mean, x_variance, x_skewness, x_kurtosis = describe(x)
print(x_nobs)
```

这个示例创建了一个数组 x，从一个均值为 5 标准差为 2 的正态分布中提取出 1000 个值。函数 mean、min、max、var 和 std 分别计算出 x 的均值、最小值、最大值、方差和标准差。同样，describe 函数可以返回 x 中的观测数量、最小值和最大值、均值和方差、以及偏斜度和峰度。

线性回归。`stats` 扩展包简化了在线性回归中估计斜率和截距的过程。除了斜率和截距，`linregress` 函数还可以返回相关系数、原假设（斜率为 0）的双侧 p 值，以及估计的标准差：

```
from numpy.random import random
from scipy import stats
x = random(20)
y = random(20)
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)
print("R-squared:", round(r_value**2, 4))
```

在这个示例中，`pring` 语句打印出相关系数的平方，以显示 R 方的值。

以上示例相对于 SciPy 中所有子包和函数来说，只是冰山之一角。要想下载或了解更多关于 SciPy 的信息，请访问 SciPy 网站 (http://docs.scipy.org/doc/scipy/reference）。

9.2.3 Scikit-Learn

Scikit-Learn 扩展包提供了估计机器学习统计模型的功能，包括回归、分类和聚集模型，还有数据处理、数据降维和模型选择等功能。Scikit-Learn 既可以处理监督式学习模型，也可以处理无监督式学习模型。监督式模型中带有因变量值或类标号，无监督式模型中则没有因变量值和类标号。Scikit-Learn 中有一组函数，可以进行各种形式的交叉验证（使用没有用来拟合模型的数据来检验模型性能），这是它与 StatsModels 的一个主要区别。

使用拟合模型的同一数据来检验模型性能是一种错误的方法，因为这样建立的模型在使用原有数据来检验时，会相当完美地重现因变量的值或类标号。根据原有数据得出的结果，这些模型的表现会非常棒，实际上，它们对建模数据进行了过拟合，在使用新数据进行预测时，往往得不到好的结果。

为了避免过拟合，估计出在新数据上也能表现出优异性能的模型，通常是将数据集分成两部分：训练集和测试集。训练集用来构建和拟合模型，测试集用来评价模型的性能。因为拟合模型的数据与评价模型性能的数据是不同的，所以可以降低过拟合的概率。多次将数据集分成两部分，用训练集训练模型，并用测试集测试模型，这个过程就称为交叉验证。

有很多方法可以用来交叉验证，最基本的方法称为 k 折交叉验证。在 k 折交叉验证中，初始数据集被划分为一个训练集和一个测试集，训练集又被划分为 k 个部分，或称 k 折（例如：5 折或 10 折）。然后，分别留下 1 折数据用来评价模型性能，使用其余 $k-1$ 折数据作为训练数据去拟合模型，此类过程共进行 k 次。这样的交叉验证过程会产生多个性能值，每折数据对应一个性能值，最终的训练集性能测量结果就是每折的性能值的均值。最后，在测试集上运行交叉验证模型，计算出模型的总体性能值。

为了表现在 Scikit-Learn 中构建统计学习模型有多么简单直接，可以使用交叉验证建立一个随机森林模型。如果你不了解随机森林模型，可以查看一下维基百科中的条目（https://en.wikipedia.org/wiki/Random_forest），获得一个整体的了解。如果想更深入地研究一下，可以参考 Trevor Hastie、Robert Tibshirani 和 Jerome Friedman 的著作《统计学习基础》，或者 Max Kuhn 和 Kjell Johnson 的著作《应用预测建模》，这两本著作都是这个领域内的绝好

教材。在 Scikit-Learn 中，通过寥寥几行代码，就可以使用交叉验证构建一个随机森林模型并对模型性能做出评价：

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import KFold
from sklearn.ensemble import RandomForestClassifier as RF

y = data_frame['Purchased?']
y_pred = y.copy()
feature_space = data_frame[numeric_columns]
X = feature_space.as_matrix().astype(np.float)
scaler = StandardScaler()
X = scaler.fit_transform(X)

kf = KFold(len(y), n_folds=5, shuffle=True, random_state=123)
for train_index, test_index in kf:
    X_train, X_test = X[train_index], X[test_index]
    y_train = y[train_index]
    clf = RF()
    clf.fit(X_train, y_train)
    y_pred[test_index] = clf.predict(X_test)

accuracy = np.mean(y == y_pred)
print "Random forest: " + "%.3f" % (accuracy)
```

前 5 行代码导入 NumPy、pandas 和 Scikit-Learn 中的 3 个组件。这 3 个组件的功能依次为：对解释变量进行中心化与比例化、执行 *k* 折交叉验证、应用随机森林分类器。

下一段代码进行以下处理：定义因变量 `y`、创建解释变量矩阵 `X`、对解释变量进行中心化和比例化。这段代码假设你已经创建了一个 pandas 数据框 `data_frame`，并且因变量数据在 `Purchased?` 列中。`copy` 函数生成了因变量的一个副本，并将其赋给变量 `y_pred`，这个变量将用来评价模型性能。下一行代码假设你已经在 `data_frame` 中创建了一个数值型变量列表，所以你可以使用列表中的变量作为解释变量集合（特征集合）。再下一行代码使用 NumPy 和 pandas 函数将特征集合转换为一个矩阵 `X`。这段中的最后 2 行代码使用 Scikit-Learn 函数创建一个 scaler 对象，并使用这个对象对解释变量进行中心化和比例化。

下一段代码使用随机森林分类器实现 *k* 折交叉验证。第一行代码使用 `KFold` 函数将数据集分成 5 份，或称 5 折，作为训练集和测试集。下一行代码是一个 `for` 循环，在每折数据之间迭代。在 `for` 循环中，对于每一折，将训练集和测试集数据赋给相应的解释变量，将训练集数据赋给因变量，初始化随机森林分类器，使用训练集数据拟合随机森林模型，然后使用模型和测试集数据为因变量估计预测值。

最后一段代码计算并报告模型的准确度。第一行代码使用 NumPy 的 `mean` 函数计算出因变量预测值等于实际初始值的平均次数。括号中的判断条件检验这两个值是否相等（即它们是否同时为 1 或者同时为 0），所以 `mean` 函数计算出一系列 1 和 0 的均值。如果所有预测值都匹配了初始数据值，那么均值为 1。如果所有预测值都不匹配初始数据值，那么均值为 0。因此，这里希望交叉验证随机森林分类器产生一个接近于 1 的均值。最后一行在屏

幕上打印出保留 3 位小数的模型准确度。

这个示例演示了在 Scikit-Learn 中使用随机森林分类器进行交叉验证的方法。除了本节中介绍的随机森林模型，Scikit-Learn 中还有很多其他类型的回归和分类模型。例如，只要将下面的 `import` 语句添加到脚本中，并且将分类器从 `clf = RF()` 修改为 `clf = SVC()`，就可以实现一个支持向量机模型：

```
from sklearn.svm import SVC
```

除了各种模型，Scikit-Learn 中还有很多用于数据预处理、数据降维和模型选择的函数。

要了解更多关于 Scikit-Learn 的信息，以及其他使用交叉验证估计模型的方法，请参考 Scikit-Learn 文档（http://scikit-learn.org/stable/index.html）。

9.2.4 更多的扩展包

除了 NumPy、SciPy 和 Scikit-Learn，根据不同类型数据分析工作的要求，你可能需要了解更多的扩展包。下面的列表相对于 Python 包索引中成千上万的扩展包来说，只是沧海一粟，列出的这些扩展包仅是作为一个推荐，希望你对此感兴趣并能实际应用：

- xarray（http://xray.readthedocs.org/en/stable/#）
  提供一个类似 pandas 的工具箱，用于多维数组分析
- SKLL（https://skll.readthedocs.org/en/latest/index.html）
  为一般的 Scikit-Learn 操作提供命令行工具
- NetworkX（https://networkx.github.io/documentation.html）
  提供创建、生长和分析复杂网络的函数
- PyMC（https://pymc-devs.github.io/pymc/index.html）
  提供实现贝叶斯统计和 MCMC（马尔科夫链蒙特卡洛）方法的函数
- NLTK（http://www.nltk.org/）
  提供用于自然语言处理的文本处理和分析工具
- Cython（http://cython.org/）
  提供一个在 Python 中生成和调用快速 C 代码的接口

以上扩展包不包含在 Python 安装程序中，你必须分别下载并安装。如想下载安装这些扩展包，可以访问 Python 包索引网站（https://pypi.python.org/pypi），或者 Unofficial Windows Binaries for Python Extension Packages 网站（http://www.lfd.uci.edu/~gohlke/pythonlibs/）。

9.3 更多的数据结构

当你结束了对本书的学习，开始使用 Python 完成各种各样的商业数据处理与分析任务时，就会越来越觉得掌握更多的数据结构是非常重要的。通过学习这些概念，你会极大地提高解决问题的能力，在面临问题时，不仅可以提出各种不同的解决方案，还可以在不同的方案之中做出权衡取舍。同时，你也可以根据实际情况恰当地选择数据结构，来更快速有效

地保存、处理和分析你的数据。

你还需要了解的其他数据结构包括栈、队列、图和树。在特定情况下，这些数据结构可以更有效地保存和检索数据，能够比列表、元组和字典更好地使用内存。

9.3.1 栈

栈是一个项目的有序集合，既可以向栈中的一端添加一个项目，又可以从同一端删除一个项目，但一次只能添加或删除一个项目。可以添加和删除项目的一端称为栈顶，相反的一端称为栈底。给定了栈顶和栈底之后，栈顶附近的项目停留在栈中的时间要少于栈底附近的项目。另外，从栈中删除项目的顺序和向栈中添加项目的顺序是相反的。这个特性称为 LIFO（last in, first out，后入先出）。

可以将栈想象为自助餐厅中的一叠盘子。要想产生一叠盘子，可以先向柜台上放一个盘子，然后向第一个盘子上面再放一个盘子，并一直重复这个操作。要想减少这叠盘子的数量，就要从顶端拿走一个盘子。你可以在任何时候增加或减少一个盘子，但是必须一直从顶端增加或减少。

有很多数据处理和分析问题适合使用栈来解决。人们使用栈来分配和访问计算机内存、保存命令行参数和函数参数、解析表达式、反转数据项、保存与回溯 URL，等等。

9.3.2 队列

队列是一个项目的有序集合，既可以向队列的一端添加项目，又可以从队列的另一端删除项目。在队列中，可以向队列后端添加项目，这样项目会依次向前移动，并在前端被移出队列。给定了队列的前端和后端，队列后端附近的项目停留在队列中的时间要少于队列前端附近的项目。这个特性称为 FIFO（first in, first out，先入先出）。

想象一下你曾经排过的一个秩序良好的队列，或者队伍。不管是在主题公园、电影院，还是在杂货店，你进入队列，排在末尾，然后向前移动直到队列最前方，这时你可以买票或接受服务，然后你就离开了队列。

有很多数据处理和分析问题适合使用队列来解决。人们使用队列来处理打印机上的打印任务、控制计算机进程等待资源、优化队列和网络流程，等等。

9.3.3 图

图是一组结点（也称为顶点）和连接结点的边的集合。边可以是有方向的，表示两个结点之间的方向，也可以是无方向的，仅表示两个结点之间的连接。边还可以具有权重，表示两个结点之间的关系，根据具体问题的情况，权重可以不同。

想象任何表示人群、地点和主题之间关系的图。例如，可以想象一下表示演员、导演和电影之间关系的图，演员、导演和电影就是图中的结点，结点之间的边表示哪个演员出演了这部电影，或者哪个导演指导了这部电影。再比如，可以将城市作为结点，结点之间的边表示从一个城市到另一个城市之间的道路。这样的边就可以使用权重表示两个城市之间的距离。

有很多数据处理和分析问题适合使用图来解决。人们使用图来表示供应商、客户和产品之间的关系，或者实体之间的关系，还使用图来表示地图，以及不同类型资源的储量和需求量。

9.3.4 树

从数据结构意义上说，树是图的一种特殊形式，它由一组层次化的结点和边组成。在树中，有一个最顶层的结点，称为根结点。根结点可以有任意数量的子结点，每个子结点也可以有任意数量的子结点。一个子结点只能有一个父节点。如果每个结点有多达两个子结点，那么这样的树就被称为二叉树。

考虑一下 HTML 文件中的元素。在 `html` 标签之中，有 `head` 标签和 `body` 标签。在 `head` 标签之中，有 `meta` 标签和 `title` 标签。在 `body` 标签之中，有 `h1`、`div`、`form` 和 `ul` 标签。如果用树结构表示这些标签的话，就可以将 `html` 标签看作根结点，它有两个子结点，`head` 标签和 `body` 标签。在表示 `head` 标签的结点下面，有 `meta` 标签和 `title` 标签。在表示 `body` 标签的结点下面，有 `h1` 标签、`div` 标签、`form` 标签和 `ul` 标签。

有很多数据处理和分析问题适合使用树来解决。人们使用树来建立计算机中的文件系统、管理层次化的数据、使信息更易于检索，以及表示句子中的短语结构等。

本节简单介绍了一些经典的数据结构。如果想学习更多如何在 Python 中使用这些数据结构的知识，可以参考 Brad Miller 和 David Ranum 的在线著作 *Problem Solving with Algorithms and Data Structures Using Python*（http://interactivepython.org/runestone/static/pythonds/index.html）。

知道这些数据结构的存在对你是很有帮助的，当你的程序运行得不是那么好的时候，可以试试这些数据结构。了解并知道在什么情况下应该使用哪种数据结构，你就有能力去解决各种规模庞大、难度系数高的问题，并能提高脚本的性能，节省处理时间，提高内存的利用效率。

9.4 从这里启航

当你开始阅读本书时，如果从来没有接触过编程，那么在跟随书中的示例进行练习时，也应该获得了很多基础的编程经验。本书先从下载安装 Python 开始，使用文本编辑器编写了一个基本的 Python 脚本，然后演示了如何在 Windows 系统和 macOS 系统中运行这个脚本。在此之后，本书开发了第一个脚本，来学习 Python 中的基本数据类型、数据结构、控制流，以及读写文本文件的方法。然后，本书演示了如何解析 CSV 文件中特定的行与列，如何解析 Excel 文件中特定的工作表、行与列，以及如何在数据库中加载、修改和输出数据。在第 3 章和第 4 章中，本书下载安装了 MySQL 和一些 Python 扩展模块。在获得了以上成功经验之后，本书在第 5 章中使用新学的编程技能完成了 3 个实际应用，并对编程能力进行了扩展。然后，在第 6 章和第 7 章中，本书从数据处理过渡到了数据可视化和统计分析。最后，在第 8 章中，本书演示了如何自动运行脚本，这样不需要在命令行中手动操作，脚本就可以定期运行了。在将要结束本书时，你可能正在思考这个问题：下一步该做些什么呢？

在自己的学习经历中，我曾经得到过一些有价值的建议，比如：确定一个你认为可以通过 Python 来改善的、重要的或者感兴趣的具体问题（或任务），然后开始着手解决这个问题，直到完成你设定的目标为止。你应该选择一个重要的或者感兴趣的问题，这样你就会对这个问题充满热情，并投入大量精力来达到你的目标。在解决问题的过程中，你会遇到很多绊脚石，走各种弯路，还可能进入死胡同，所以你选择的问题应该足够重要，以使你能够坚持编写、调试、修改代码，克服各种困难，直至使代码正确运行。还有，在选择具体的问题和任务时，你要定义清楚需要使用代码去做什么。例如，你的问题可能是有太多需要手工处理的文件，所以你需要解决的就是如何用 Python 来处理这些文件。或者，你可能正在负责一项具体的数据处理与分析任务，你认为使用 Python 可以自动地完成任务，并能提高效率和一致性。只要你确认了具体的问题和任务，就可以更容易地将问题分解为一个个子环节，来逐步解决并完成你的目标。

只要你选定了具体问题或任务，并划分出了解决问题的各个环节，你就掌握了解决问题的主动权。相对于一次性解决整个问题来说，每次解决一个环节更加容易。这里我要引用一句格言："如何吃掉一条鲸鱼？当然要一口一口地吃。"每次解决一个环节的优势在于，对于这个环节，非常可能已经有人遇到过类似的问题，他（她）已经解决了这个问题，并在网络上或某本书中给出了解决方案。

互联网是你的朋友，特别是遇到编程问题时。本书已经介绍了如何读取 CSV 文件和 Excel 文件，但是当你需要读取其他类型的文件，比如 JSON 文件或 HTML 文件时，应该怎么办呢？可以打开浏览器，在搜索栏中输入"python read json file examples"，看看其他人是如何用 Python 读取 JSON 文件的。对于你将问题分解成的各个环节，这个方法也同样适用。当你将问题具体化到一个很小的范围内时，搜索在线资源将非常有效。此外，除了在线资源，还有很多 Python 书籍和培训资料，其中有很多有用的代码片段和示例。你可以在网上找到很多免费的 PDF 格式的 Python 书籍，在图书馆中，也有很多这样的书籍。我的观点是，你不用重新发明轮子。对于你的整体问题和任务中的每个小环节，应该先看看本书、互联网或者其他资源中是否已经有了解决问题的方法，然后对这些方法进行一些修改和调试，直至问题解决。在解决了每个环节之后，你就可以得到一个 Python 脚本，解决你的具体问题或任务。这就是你所追求的激动人心的一刻：轻轻地按下一个键，经过数天或数周辛苦劳动完成的代码开始工作，按照你的指令运行，完美地解决了你的问题或任务。这种感觉激动人心，使人充满力量。一旦你意识到可以高效地完成那些枯燥无味、浪费时间、容易出错的手工不可能完成的任务，就会感到一种激情流遍全身，迫不及待地希望使用 Python 去完成更多的问题和任务。这就是我希望你去做的事情，从这里启航，选择一个重要的问题或任务去攻克，直到你的代码正确运行，到达胜利的彼岸。

附录 A

下载指南

A.1　下载Python 3

A.1.1　Windows

(1) 访问 https://www.python.org/downloads（下载页面会检测你的操作系统，并建议你安装 Windows 版本）。
(2) 点击“Download Python 3.4.3”（或者 Python 3 的最新版本，在本书出版之后，应该已经更新了）。
(3) 点击“Windows x86 MSI installer”，保存或运行 MSI 安装程序。
(4) 双击下载的 python-3.4.3.msi 安装程序，启动 Python 安装。
(5) 选择“Install for all users”或“Install just for me”，然后点击“Next”。
(6) 使用默认目标目录（C:\Python34\），然后点击“Next”。
(7) 对所有后续步骤，不修改默认设置，一直点击“Next”。

这样 Python 就安装好了。

Python 安装完成之后，执行以下操作。

(1) 点击“开始”按钮。
(2) 点击“控制面板”。
(3) 点击“系统和安全”。
(4) 点击“系统”。
(5) 点击“高级系统设置”。
(6) 点击“高级”标签页。
(7) 点击“环境变量”。

(8) 在“系统变量”选项中，向下滚动并单击“Path”变量。
(9) 点击“编辑”。
(10) 在“变量值”域中，检查并确认“C:\Python34\”在变量列表中。

如果没在列表中，那么在列表末尾输入“;C:\Python34\”，将路径添加到列表中（分号用来分隔各个独立的路径）。

(11) 点击“确定”，保存对路径系统变量所做的修改。
(12) 点击“确定”，退出环境变量窗口。
(13) 点击“确定”，退出系统属性窗口。
(14) 点击打开资源管理器。
(15) 双击 C: 磁盘驱动器。
(16) 双击 Python34 文件夹。
(17) 双击 python 应用程序。

如果打开了 Python Shell 窗口，那么就说明 Python 安装成功了。

A.1.2 macOS

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 输入以下命令，然后按回车键：

```
which python
```

如果你看到像 /usr/bin/python/ 或 /usr/local/bin/python 这样的路径显示在终端窗口中，那么 Python 就已经安装好了，你可以进行下面的工作了。

如果你没有看到像 /usr/bin/python/ 或 /usr/local/bin/python 这样的路径，那么就按照下面的指示来安装 Python。

(1) 访问 https://www.python.org/downloads（下载页面会检测你的操作系统，并建议你安装 Mac OS 版本）。
(2) 点击“Download Python 3.4.3”（或者 Python 3 的最新版本，在本书出版之后，应该已经更新了）。
(3) 点击“Save File”，下载 python-3.4.3-macosx10.6.pkg。
(4) 双击 python-3.4.3-macosx10.6.pkg 启动 Python 安装。
(5) 点击“Continue”，离开欢迎界面。
(6) 点击“Continue”，离开重要信息提示界面。
(7) 点击“Agree/Continue”，离开软件许可协议界面。
(8) 点击“Install”，在默认目录下安装 Python。
(9) 对于所有后续步骤，不修改默认设置，一直点击“Continue”。

这样 Python 就安装好了。

(10) 点击“Applications”打开应用程序窗口。
(11) 点击“iTerm”打开终端窗口。

(12) 输入以下命令，然后按回车键：

```
which python
```

你应该可以看到像 /usr/bin/python/ 或 /usr/local/bin/python 这样的路径，这说明 Python 已经安装好了，你可以进行下面的工作了。

A.2　下载xlrd扩展包

A.2.1　Windows

方法1

在进行以下步骤之前，你必须先安装好 Python 3。

(1) 打开命令行窗口。
(2) 输入以下命令，然后按回车键：

```
python -m pip install xlrd
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `xlrd` 扩展包已经安装好了。

要确认 `xlrd` 安装正确，执行以下步骤。

(1) 点击打开资源管理器。
(2) 双击 C: 磁盘驱动器。
(3) 双击 Python34 文件夹。
(4) 双击 python 应用程序。
(5) 当 Python Shell 窗口打开后，输入以下命令，然后按回车键：

```
import xlrd
```

如果你没有接收到任何错误信息，那么就说明 `xlrd` 安装正确，你可以进行下面的工作了。

方法2

(1) 访问 https://pypi.python.org/pypi/xlrd。
(2) 点击绿色的“Downloads”按钮，下载 `xlrd` 的最新版本。
(3) 在下载的程序文件上点击鼠标右键，选择“在文件夹中显示”。
(4) 将文件解压到 Downloads 文件夹中。
(5) 双击解压后的 xlrd-0.9.3.tar 文件夹，进入文件夹。
(6) 点击并复制解压后的 xlrd-0.9.3 文件夹。
(7) 回到 Downloads 文件夹，将解压后的 xlrd-0.9.3 文件夹粘贴到这个文件夹。
(8) 打开命令行窗口。
(9) 输入以下命令，然后按回车键，转到 Downloads 文件夹：

```
/cd Downloads
```

(10) 输入以下命令，然后按回车键，转到 xlrd-0.9.3 文件夹：

```
cd xlrd-0.9.3
```

(11) 现在已经来到了 xlrd-0.9.3 文件夹中，输入以下命令，然后按回车键：

```
python setup.py install
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `xlrd` 扩展包已经安装好了。

要确认 `xlrd` 安装正确，执行以下步骤。

(1) 点击打开资源管理器。
(2) 双击 C: 磁盘驱动器。
(3) 双击 Python34 文件夹。
(4) 双击 python 应用程序。
(5) 在 Python Shell 窗口打开后，输入以下命令，然后按回车键：

```
import xlrd
```

如果你没有接收到任何错误信息，那么就说明 `xlrd` 安装正确，你可以进行下面的工作了。

A.2.2　macOS

方法1

在进行以下步骤之前，你必须先安装好 Python 3。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 输入以下命令，然后按回车键：

```
python -m pip install xlrd
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `xlrd` 扩展包已经安装好了。

> 如果你收到了错误信息，请试着输入以下命令，然后按回车键：
>
> ```
> sudo python -m pip install xlrd
> ```
>
> 你会被要求输入登录进计算机的密码。输入密码（不会显示在屏幕上），然后按回车键。

要确认 `xlrd` 安装正确，执行以下步骤。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 在终端窗口中打开 Python 解释器，输入以下命令，然后按回车键：

```
python
```

(4) Python 解释器打开后，输入以下命令，然后按回车键：

```
import xlrd
```

如果你没有接收到任何错误信息，那么就说明 `xlrd` 安装正确，你可以进行下面的工作了。

方法2

(1) 访问 https://pypi.python.org/pypi/xlrd。
(2) 点击绿色的“Downloads”按钮，转到可下载文件。
(3) 点击“xlrd-0.9.3.tar.gz”（或最新版本，在本书出版时，应该已经更新了），将压缩文件保存到 Downloads 文件夹。
(4) 双击下载的文件，将文件解压到 Downloads 文件夹。

> 如果在解压文件时遇到问题，也可以在终端窗口中解压文件。在终端窗口中输入以下命令，然后按回车键，转到 Downloads 文件夹：
>
> ```
> cd Downloads
> ```
>
> 接下来，开始解压文件，输入以下命令，然后按回车键：
>
> ```
> tar -zxvf xlrd-0.9.3.tar.gz
> ```
>
> 解压后的文件夹 xlrd-0.9.3 会出现在 Downloads 文件夹中。

(5) 点击“Applications”打开应用程序窗口。
(6) 点击“iTerm”打开终端窗口。
(7) 输入以下命令，然后按回车键，转到 Downloads 文件夹：

```
cd Downloads/
```

(8) 输入以下命令，然后按回车键，转到 xlrd-0.9.3 文件夹：

```
cd xlrd-0.9.3/
```

(9) 现在已经来到了 xlrd-0.9.3 文件夹中，输入以下命令，然后按回车键：

```
python setup.py install
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `xlrd` 扩展包已经安装好了。

> 如果你收到了错误信息，请试着输入以下命令，然后按回车键：
>
> ```
> sudo python setup.py install
> ```
>
> 你会被要求输入登录进计算机的密码。输入密码（不会显示在屏幕上），然后按回车键。

要确认 `xlrd` 安装正确，执行以下步骤。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 在终端窗口中打开 Python 解释器，输入以下命令，然后按回车键：

```
python
```

(4) Python 解释器打开后，输入以下命令，然后按回车键：

```
import xlrd
```

如果你没有接收到任何错误信息，那么就说明 `xlrd` 安装正确，你可以进行下面的工作了。

A.3　下载MySQL数据库服务器

A.3.1　Windows

(1) 访问 http://dev.mysql.com/downloads/mysql。
(2) 点击“MySQL Community Server”。
(3) 点击“Windows (x86, 32-bit), MySQL Installer MSI”旁边的“Download”按钮。
(4) 点击“No thanks, just start my download.”。
(5) 下载结束后，点击下载的安装程序进行安装。
(6) 按照安装程序中的指示完成安装。

A.3.2　macOS

(1) 访问 http://dev.mysql.com/downloads/mysql。
(2) 点击“MySQL Community Server”。
(3) 点击“Mac OS X 10.11 (x86, 64-bit), DMG Archive.”旁边的“Download”按钮。

> 一定要选择 .dmg 文件，这个才是安装文件。

(4) 点击“No thanks, just start my download.”。
(5) 下载结束后，点击下载的安装程序进行安装。
(6) 按照安装程序中的指示完成安装。

A.3.3　启动MySQL

我不想骗你，这一步很可能会出问题。MySQL 参考手册（http://dev.mysql.com/doc/refman/5.7/en）中关于如何安装的部分对此进行了详细介绍，但是，按照手册中的步骤启动 MySQL，总是会有一两条错误信息需要你使用 Google 去解决。

A.4　下载`mysqlclient`（Python 3.x）/MySQL-python（Python 2.x）

在进行以下步骤之前，你应该先下载安装 MySQL。这个 Python 扩展包的安装程序在进行安装时会检查 MySQL 配置文件，如果没有发现 MySQL，安装就会失败。

A.4.1　Windows

方法1

在进行以下步骤之前，你必须先安装好 Python 3。

(1) 打开命令行窗口。
(2) 输入以下命令，然后按回车键：

```
python -m pip install mysqlclient
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `mysqlclient` 扩展包已经安装好了。

要确认 `mysqlclient` 安装正确，执行以下步骤。

(1) 点击打开资源管理器。
(2) 双击 C: 磁盘驱动器。
(3) 双击 Python34 文件夹。
(4) 双击 python 应用程序。
(5) 在 Python Shell 窗口打开后，输入以下命令，然后按回车键：

```
import MySQLdb
```

如果你没有接收到任何错误信息，那么就说明 `mysqlclient` 安装正确，你可以进行下面的工作了。

方法2

(1) 访问 http://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient。
(2) 点击与你的 Python 版本（3.x 或 2.x）和操作系统版本（32-bit 或 64-bit）对应的 `mysqlclient` 版本链接。
(3) 将文件保存至 Downloads 文件夹。

> 要确定所需的版本，你可以打开一个命令行窗口，输入 `python` 并按回车键，打开 Python 解释器，然后查看屏幕上方的头信息。你会看到像“Python 3.4.3(32-bit)”这样的信息，这说明你应该选择 32 位的 Python3.4 使用的版本。在写作本书的时候，链接应该为：mysqlclient-1.3.6-cp34-none-win32.whl。如果你安装了不同版本的 Python 或不同版本的操作系统，那么你需要按照你的版本选择链接。

(4) 打开命令行窗口。
(5) 输入以下命令，然后按回车键：

```
cd Downloads
```

(6) 输入以下命令（可以按照实际情况使用不同的文件名），然后按回车键：

```
python -m pip install mysqlclient-1.3.6-cp34-none-win32.whl
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `mysqlclient` 扩展包已经安装好了。

要确认 `mysqlclient` 安装正确，执行以下步骤。

(1) 点击打开资源管理器。
(2) 双击 C: 磁盘驱动器。

(3) 双击 Python34 文件夹。
(4) 双击 python 应用程序。
(5) 在 Python Shell 窗口打开后，输入以下命令，然后按回车键：

```
import mysqlclient
```

如果你没有接收到任何错误信息，那么就说明 mysqlclient 安装正确，你可以进行下面的工作了。

A.4.2 macOS

方法1

在进行以下步骤之前，你必须先安装好 Python 3。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 输入以下命令，然后按回车键：

```
python -m pip install mysqlclient
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 mysqlclient 扩展包已经安装好了。

> 如果你收到了错误信息，请试着输入以下命令，然后按回车键：
>
> ```
> sudo python -m pip install mysqlclient
> ```
>
> 你会被要求输入登录进计算机的密码。输入密码（不会显示在屏幕上），然后按回车键。

要确认 mysqlclient 安装正确，执行以下步骤。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 在终端窗口中打开 Python 解释器，输入以下命令，然后按回车键：

```
python
```

(4) Python 解释器打开后，输入以下命令，然后按回车键：

```
import mysqlclient
```

如果你没有接收到任何错误信息，那么就说明 mysqlclient 安装正确，你可以进行下面的工作了。

方法2

(1) 访问 https://pypi.python.org/pypi/mysqlclient。
(2) 点击绿色的“Downloads”按钮，转到可下载文件。
(3) 点击“mysqlclient-1.3.6.tar.gz”（或最新版本，在本书出版时，应该已经更新了），将压缩文件保存到 Downloads 文件夹。

(4) 双击下载的文件，将文件解压到 Downloads 文件夹。

> 如果在解压文件时遇到问题，也可以在终端窗口中解压文件。在终端窗口中输入以下命令，然后按回车键，转到 Downloads 文件夹：
>
> ```
> cd Downloads
> ```
>
> 接下来，开始解压文件，输入以下命令，然后按回车键：
>
> ```
> tar -zxvf mysqlclient-1.3.6.tar.gz
> ```
>
> 解压后的文件夹 mysqlclient-1.3.6 会出现在 Downloads 文件夹中。

(5) 点击“Applications”打开应用程序窗口。
(6) 点击“iTerm”打开终端窗口。
(7) 输入以下命令，然后按回车键，转到 Downloads 文件夹：

```
cd Downloads/
```

(8) 输入以下命令，然后按回车键，转到 mysqlclient-1.3.6 文件夹：

```
cd mysqlclient-1.3.6/
```

(9) 现在已经来到了 mysqlclient-1.3.6 文件夹中，输入以下命令，然后按回车键：

```
python setup.py install
```

按下回车键之后，你会看到命令行窗口中的输出信息，表示 `mysqlclient` 扩展包已经安装好了。

> 如果你收到了错误信息，请试着输入以下命令，然后按回车键：
>
> ```
> sudo python setup.py install
> ```
>
> 你会被要求输入登录进计算机的密码。输入密码（不会显示在屏幕上），然后按回车键。

要确认 `mysqlclient` 安装正确，执行以下步骤。

(1) 点击“Applications”打开应用程序窗口。
(2) 点击“iTerm”打开终端窗口。
(3) 在终端窗口中打开 Python 解释器，输入以下命令，然后按回车键：

```
python
```

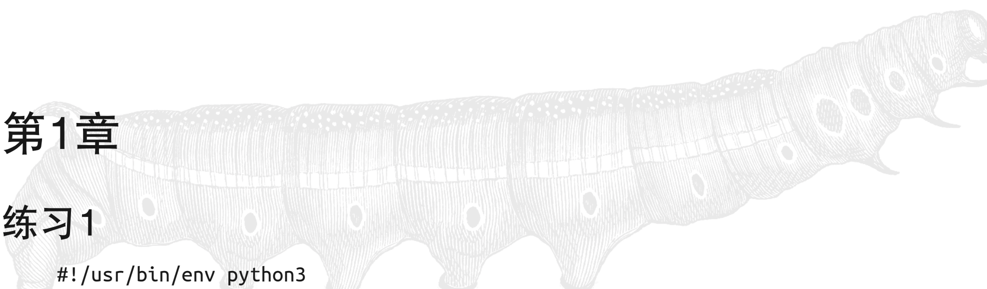
(4) Python 解释器打开后，输入以下命令，然后按回车键：

```
import mysqlclient
```

如果你没有接收到任何错误信息，那么就说明 `mysqlclient` 安装正确，你可以进行下面的工作了。

附录B

练习答案



第1章

练习1

```
#!/usr/bin/env python3
farm_animals = ['cow','pig','horse']
domestic_animals = ['dog','cat','gold fish']
zoo_animals = ['lion','elephant','gorilla']
animals = farm_animals + domestic_animals + zoo_animals
for index_value in range(len(animals)):
    print("{0:d}: {1!s}".format(index_value, animals[index_value]))
```

练习2

```
#!/usr/bin/env python3
animals_dictionary = {}
animals_list = ['cow','pig','horse']
other_list = [4567,[4,'turn',7,'left'],'Animals are great.']
for index_value in range(len(animals_list)):
    if animals_list[index_value] not in animals_dictionary:
        animals_dictionary[animals_list[index_value]] = other_list[index_value]
for key, value in animals_dictionary.items():
    print("{0!s}: {1}".format(key, value))
```

练习3

```
#!/usr/bin/env python3
```

```python
list_of_lists = [['cow','pig','horse'], ['dog','cat','gold fish'],\
['lion','elephant','gorilla']]
for animal_list in list_of_lists:
    max_index = len(animal_list)
    output = ''
    for index in range(len(animal_list)):
        if index < (max_index-1):
            output += str(animal_list[index])+','
        else:
            output += str(animal_list[index])+'\n'
print(output)
```

作者介绍

Clinton W. Brownley 博士，Facebook 数据科学家，负责数据流水线、统计建模和数据可视化项目，并为大型基础设施建设提供数据驱动的决策建议。Clinton 是美国统计协会旧金山湾区分会的前负责人，还是运筹学与管理科学协会实践部委员。Clinton 具有卡内基梅隆大学和美利坚大学的学位。

封面介绍

本书封面上的动物是一只夹竹桃蛾幼虫（拉丁文名称为 Syntomeida epilais）。

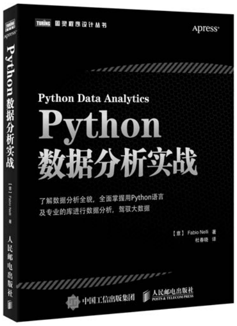
夹竹桃蛾幼虫是橙色的，身上黑毛丛生。它们主要食用夹竹桃。夹竹桃是一种常绿灌木，也是一种常见有毒的园艺植物。夹竹桃蛾幼虫对夹竹桃的毒性免疫，并通过取食和消化夹竹桃，来使任何想要吃掉它们的鸟类和哺乳动物中毒。当夹竹桃在 17 世纪被西班牙人引入佛罗里达时，夹竹桃蛾就已经生活在佛罗里达了。它们寄生在本地的葡萄藤上，但是当夹竹桃大量繁殖、更容易找到时，这种蛾子就适应了新的植物，大量寄居在夹竹桃上，因而被称为夹竹桃蛾。

成年夹竹桃蛾简直是美轮美奂：它的身体和翅膀是闪亮多变的蓝色，带有白色小圆点，腹部末端为亮红色。这种蛾子白天活动积极，像黄蜂一样飞行较慢。雌蛾栖息在夹竹桃叶子上，发射出一种超声波信号，吸引远处的雄蛾。当雄蛾与雌蛾彼此相距几米时，它们就开始一场求偶二重唱，直至交尾为止。交尾一般发生在黎明之前的两到三个小时。交尾过后，雌蛾会在夹竹桃叶子的底面产卵。卵块中有 12~75 只受精卵。受精卵孵化后，幼虫群居在一起，以粗叶脉和细叶脉之间的植物组织为食，直至叶子枯萎。这种对叶子的损害不会使植物死亡，但确实会使植物容易感染其他疾病。

O’Reilly 图书封面上的很多动物都是濒危物种，它们对世界都很重要。如果你想为保护动物做些贡献，请访问 animals.oreilly.com。

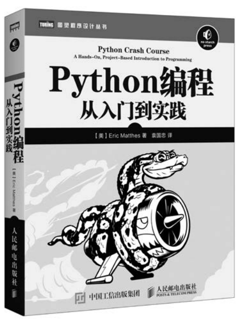
封面照片来自 *Wood’s Illustrated Natural History*。

延 展 阅 读



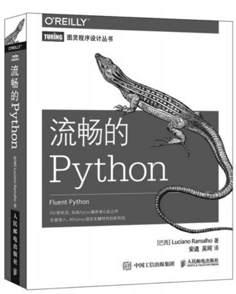
了解数据分析全貌，全面掌握用Python语言及专业的库进行数据分析，驾驭大数据

书号：978-7-115-43220-9
定价：59.00 元



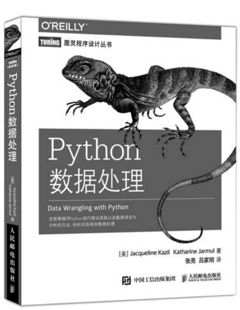
- Amazon编程入门类榜首图书
- 从基本概念到完整项目开发，帮助零基础读者迅速掌握Python编程

书号：978-7-115-42802-8
定价：89.00 元



- PSF研究员、知名PyCon演讲者心血之作
- 全面深入，对Python语言关键特性剖析到位

书号：978-7-115-45415-7
定价：139.00 元



全面掌握用Python进行爬虫抓取以及数据清洗与分析的方法，轻松实现高效数据处理

书号：978-7-115-45919-0
定价：99.00 元

O'REILLY®

Python数据分析基础

想深入应用手中的数据？还是想在上千份文件中重复同样的分析过程？没有编程经验的非程序员们如何能在最短的时间内学会用当今炙手可热的Python语言进行数据分析？

来自Facebook的数据专家Clinton Brownley可以帮您解决上述问题。在他的这本书里，读者将能掌握基本Python编程方法，学会编写出处理电子表格和数据库中的数据的脚本，并了解使用Python模块来解析文件、分组数据和生成统计量的方法。

- **学习基础语法，创建并运行自己的Python脚本**
- **读取和解析CSV文件**
- **读取多个Excel工作表和工作簿**
- **执行数据库操作**
- **搜索特定记录、分组数据和解析文本文件**
- **建立统计图并绘图**
- **生成描述性统计量并估计回归模型和分类模型**
- **在Windows和Mac环境下按计划自动运行脚本**

Clinton W. Brownley博士，Facebook数据科学家，负责大数据流水线、统计建模和数据可视化项目，并为大型基础设施建设提供数据驱动的决策建议。

“这本书对于那些使用数据的Python新手来说，是非常有用的学习资源。它的教学风格和附带的例子会帮助用户尽快熟悉Python语言、编程环境和Python生态系统中最常用的几个软件包。”

——Wes McKinney
pandas库之父

DATABASES

封面设计：Karen Montgomery　**张健**

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　**计算机 / 数据分析**

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

ISBN 978-7-115-46335-7
9 787115 463357 >

ISBN 978-7-115-46335-7
定价：69.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks